12-2013

Surface Shape Perception in Volumetric Stereo Displays

Meng Zhu
Clemson University, meng@g.clemson.edu

Surface Shape Perception in Volumetric Stereo Displays

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Meng Zhu
December 2013

Accepted by:
Dr. Donald H. House, Committee Chair
Dr. Robert M. Geist III
Dr. Joshua A. Levine
Dr. Brian A. Malloy

Abstract

In complex volume visualization applications, understanding the displayed objects and their spatial relationships is challenging for several reasons. One of the most important obstacles is that these objects can be translucent and can overlap spatially, making it difficult to understand their spatial structures. However, in many applications, for example medical visualization, it is crucial to have an accurate understanding of the spatial relationships among objects. The addition of visual cues has the potential to help human perception in these visualization tasks. Descriptive line elements, in particular, have been found to be effective in conveying shape information in surface-based graphics as they sparsely cover a geometrical surface, consistently following the geometry. We present two approaches to apply such line elements to a volume rendering process and to verify their effectiveness in volume-based graphics. This thesis reviews our progress to date in this area and discusses its effects and limitations. Specifically, it examines the volume renderer implementation that formed the foundation of this research, the design of the pilot study conducted to investigate the effectiveness of this technique, the results obtained. It further discusses improvements designed to address the issues revealed by the statistical analysis. The improved approach is able to handle visualization targets with general shapes, thus making it more appropriate to real visualization applications involving complex objects.

Dedication

This work is dedicated to my parents Jiajing Zhu and Zhiling Xu, and my wife Dr. Yan Fu. Their encouragement fortified my determination to pursue a PhD degree, without which I would not have been brave enough to embark upon such a challenging journey. Their trust in me to achieve what I strive for has always been my most valuable weapon to conquer difficulties along my way. With their unconditional support and endless patience, they have taught me what family means to each other. The love that they showed me will always light my way and remind me of what is truly important in life.

Acknowledgments

There are several people deserving my deep gratitude for their support over the past years.

First of all, I would like to thank my advisor Dr. Donald House, not only for introducing me to the fascinating field of computer graphics, his excellent education and technical guidance, but also for his patience and support to allow me to grow in a productive environment full of competent friends.

I would also like to thank Dr. Joshua Levine for always willing to share his enormous expertise and open to discussions on interesting geometry and topology topics.

Further more, I would like to thank Dr. Robert Geist and Dr. Brian Malloy for their insightful instructions on programming problems related to my research.

Apart from educating me state-of-the-art special effects techniques, I want to thank Dr. Jerry Tessendorf for connecting me to industrial research to broaden my horizon.

I would like to express my appreciation to all members of the SAVAGE Lab at Clemson University, past and present. Special thanks go to Jonathon Cox for his kind heart and being always ready to help, to Brandon Pelfrey for his challenging mathematical questions that helped me keep a sharp mind, to Dr. Andrew Duchowski and Rui Wang for sharing their statistical knowledge, and to Le Liu for his interesting questions on volume rendering and programming.

Table of Contents

List of Tables

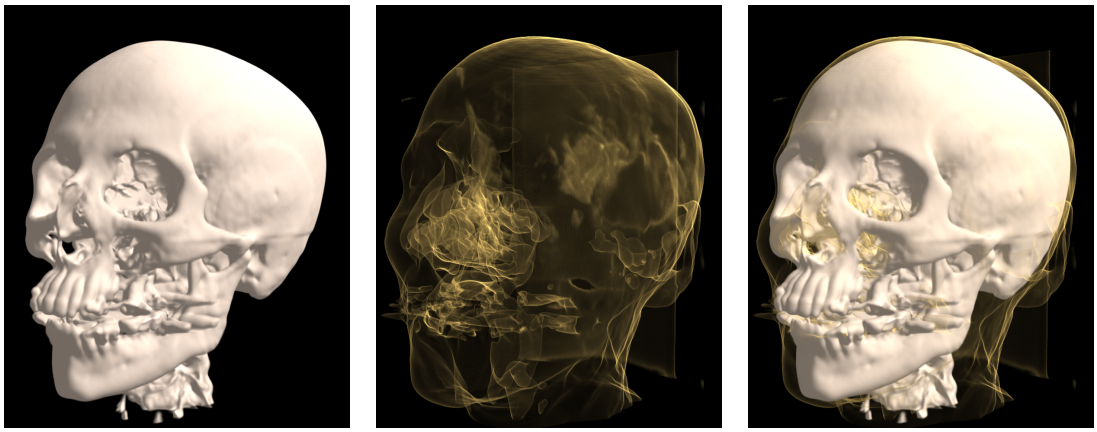
List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Volume rendering is a set of techniques particularly suited for visualizing volume data. Currently, many volume datasets are generated by various scanning technologies, simulations or artistic designs, making volume rendering an indispensable visualization technique in many fields. Figure 1.1.1 provides an example of the volume rendering of a human head dataset obtained through MRI scans.



(a) Skull (b) Skin (c) Overlapped Rendering

Figure 1.1.1: Volume Rendering of a Human Head

When volume rendering is applied to complex datasets, however, several problems are encountered, one being that in an overlapped visualization environment, traditional volume rendering

techniques tend to create visual confusion, which interferes with the accurate perception of shapes and spatial relationships. For example, when the human skull in Figure 1.1.1a and the skin in Figure 1.1.1b are viewed together in Figure 1.1.1c, it is almost impossible to visually comprehend the structure of the skin where it overlaps with the skull in the perspective projection. Specifically, the rendering in Figure 1.1.1b reveals many features in the nasal area that are not readily visible in Figure 1.1.1c. This issue is important because for many tasks the goal of visualization is to have both surfaces visually understandable. This problem cannot be corrected by simply adjusting opacities because increasing the visibility of one layer means decreasing that of the other, indicating other types of visual cues are needed to convey the structural information of the objects displayed.

This problem of visual confusion in an overlapped visualization environment has long been identified and studied by computer graphics specialists for surface-based visualization applications. One solution, adding auxiliary line texture onto surfaces, has been found to be effective in improving the human perception of the enhanced surfaces. Figure 1.1.2 shows an example of this line texture enhancement technique being applied to terrain-like surfaces. The structure of both the top and bottom surfaces are visible because of the well-designed grid-like line textures, which have been found to be one of the most effective patterns for shape perception enhancement. However, this technique

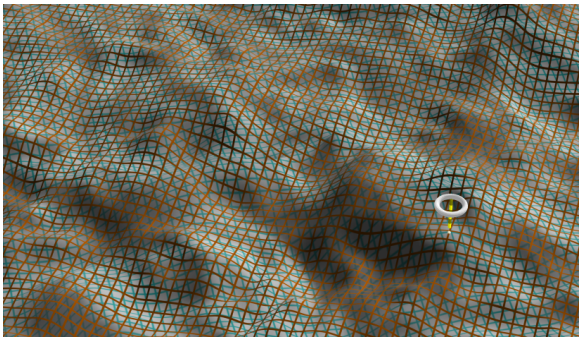


Figure 1.1.2: Line Texture Enhanced Overlapped Surfaces Rendering

has not been extended to volume rendering to enhance visualization applications involving layered surfaces. To address this situation, the goal of this work is to integrate line textures into volume visualization. Specifically we have worked in three main directions:

- First, as the basic case, we examined what perceptual gains can be made by applying line texture to opaque surfaces in a volume rendering environment. This addresses a gap in the field and provides a baseline for evaluating perceptual improvements obtained in layered surface

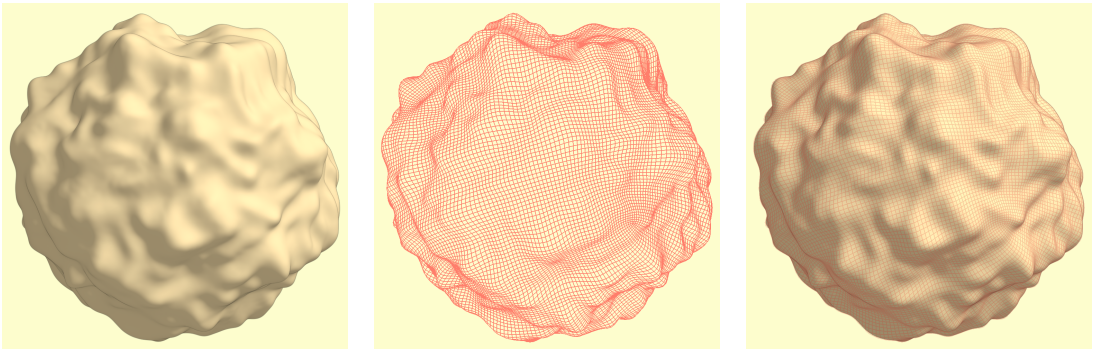
experiments.

- Second, we have conducted an empirical study to determine under what conditions line textures improve perception of surface shape in volume rendering of layered surfaces.
- Third, a texturing technique that works with volume datasets and integrates cleanly into the volume rendering architecture was developed.

1.2 Research Approach and Contributions

These objectives were achieved in two major phases. In the first part, a controlled user study was conducted, including the implementation of an interactive volume renderer, the design of irregular shapes sharing specific statistical properties and the execution of the experiments and the data analyses. The second phase improved on the results obtained from the user study as well as designed a general method that works with complex surfaces usually encountered in volume datasets provided such surfaces can be expressed as triangle meshes of 2-manifold topology.

Initially, carefully designed organic-looking shapes with statistical properties to allow user studies to be conducted in a controlled manner are developed. Specifically, the shapes are produced by randomly displacing a spherical surface with Gabor shaped bumps of varying parameters to create irregular shapes simulating complex shapes in real volume visualization applications. They

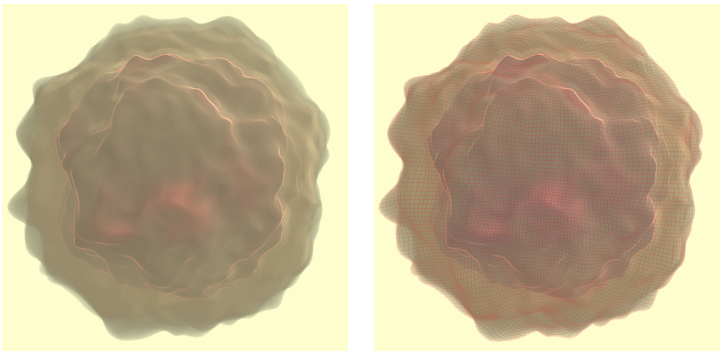


(a) Irregular Shape (b) Line Texture (c) Hybrid Rendering

Figure 1.2.1: Designed Irregular Shapes and Rendering

are converted into a volumetric format that is used for volume rendering as illustrated in Figure 1.2.1a and into quadrilateral meshes that are used to produce the line texture illustrated in Figure 1.2.1b. Figure 1.2.1c shows the combined rendering.

This approach can be applied to one shape, as shown in Figure 1.2.1c, or it can be used on two, the latter requiring an additional step merging the volume data produced for the two shapes into one, which is then used for volume rendering. The two line textures are rendered separately and then combined with the volume rendering through compositing to provide the necessary conditions for studying the overlapping visualization. Figure 1.2.2 shows a rendering involving two random surfaces, comparing them with and without line texture applied. Even at a small resolution, it can be seen that the shapes in the overlapped region are more visible when line texture is present. To



(a) Irregular Shape (b) Hybrid Rendering

Figure 1.2.2: Irregular Shapes for Overlapped Visualization Experiments

verify the effectiveness of line textures, user studies have been conducted in a stereoscopic display environment including two high resolution monitors in a Wheatstone [83] stereoscope arrangement. The reason for utilizing this display is to provide the best visual conditions for the participants as it gives additional depth clues and the high resolution gives a better visualization of the rendering. The resulting user studies and data analyses have shown that the line texture technique works for volume visualization.

However, it has a few issues, the major one being that on certain places of the display window, the line texture looks almost flat under the perspective projection. This probably occurs because the method used for applying the texture is a projection method in principle. Therefore, when viewed from a direction parallel to the original projection direction, the line texture loses its effectiveness in revealing shape. Lines having geometrical properties intrinsic to the surface being studied, such as those following principal curvature directions, may help. Since principal curvature directions are orthogonal, these lines also present a grid-like structure. Nonetheless, principal curvature directions have their own problems, thus not entirely suitable for our purpose

of generating grid-like line patterns for various visualization targets. However, they can be used to guide the generation of such a structure. In particular, important locations of an object can be identified through quantitative analysis. The principal curvature directions at these locations can then be utilized to generate a grid-like line texture heavily affected but not restricted to the principal curvature direction field, a process achieved in three steps. The first step spreads the directional information from the important locations to the entire visualization object boundary, deriving a globally consistent cross field. The second step uses the cross field to compute a 2D parameterization that optimally follows the cross field under the least square error. The last step maps the integer isogrid of the parametric space into the original model space, effectively creating two sets of lines that are mostly orthogonal and evenly distributed across the target object boundary, thus presenting a grid-like line structure fully covering the object.

To date, this research has included:

- Design of organic-looking shapes with the necessary statistical properties to evaluate the technique.
- Development of a projection-based method producing the required line pattern to be added onto these shapes.
- Implementation of a graphics hardware accelerated volume renderer that is able to deliver interactive frame rates for the targeted datasets and incorporate the line texture directly.
- Setting up a testbed and necessary analysis methods to evaluate the effectiveness of the line texture.
- Completion of two perceptual experiments.
- Design of a general method that generates grid-like line patterns for irregular shapes.

1.3 Thesis Organization

Chapter 2 discusses relevant theories, concepts and techniques that are used in this work, including volume data, volume rendering and various line elements of geometric models. Chapter 3 discusses the preliminary study, specially how the shapes were designed, the user study conducted and the data analyzed. Chapter 4 presents the work based on the results of the user study, in

particular, the method to identify important surface features, the generation of a globally consistent cross field, the parameterization based on such field and finally the production of a grid-like line structure. Chapter 5 discusses the effects and limitations of the proposed approach, providing directions for future research. Chapter 6 concludes this work.

Chapter 2

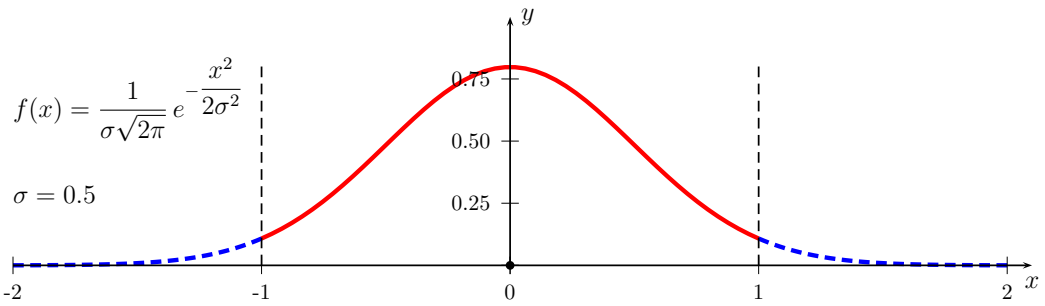
Background

This chapter explains volume rendering including its theoretical support, existing methods and applications. It also examines the research on line texture generation methods and their applications on polygonal surfaces before introducing the idea of incorporating line texture in a volume renderer directly.

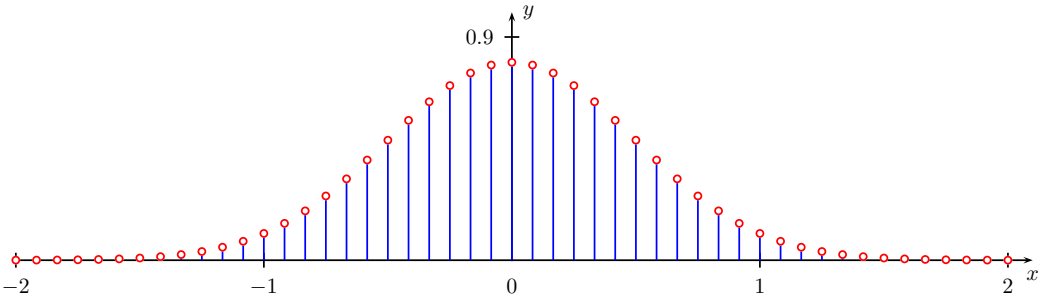
2.1 Volume Data

The volume rendering method simulates light interaction with a volumetric medium, which can be described by a general function, the domain of which is 3D space. For example, the function $f(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r$ describes a 3D sphere, mapping spatial positions to signed distances from the zero contour, which is the spherical surface of radius r centered at the origin. However, many interesting natural phenomena cannot easily be described by analytical functions. For volume rendering to be a practical technique, there must be a more general way to describe the distribution of an quantity throughout a given spatial domain.

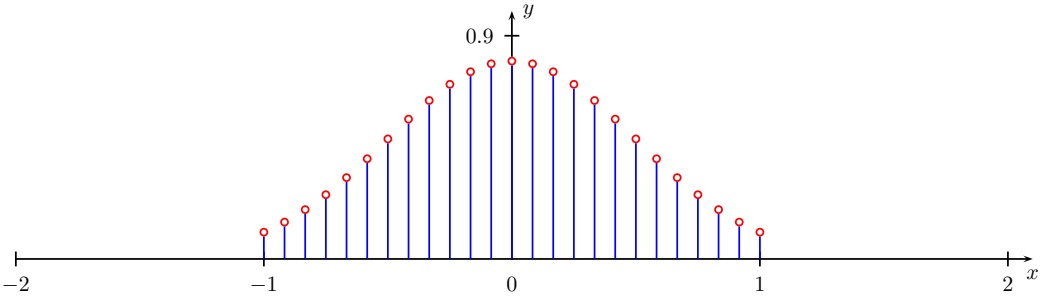
Sampling theory [69], where an arbitrary mathematical function can be represented by a number of samples living in the same space, provides the appropriate representation. As long as the sampling frequency is above the Nyquist frequency [68, 76], the original mathematical function can be unambiguously reconstructed from the set of samples. The Nyquist frequency ν_N for a signal $f(t)$ is defined as twice the maximum frequency ν_M of the signal. The maximum frequency is intuitively the upper bound value in the frequency domain of the signal, beyond which the Fourier transform



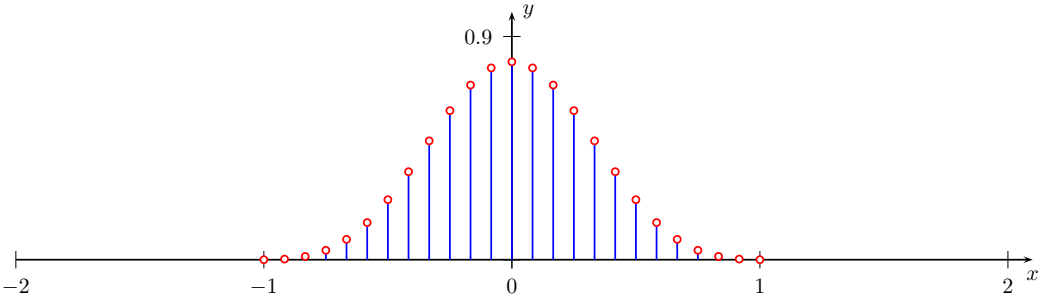
(a) Bounded



(b) Sampled



(c) Bounded and Sampled



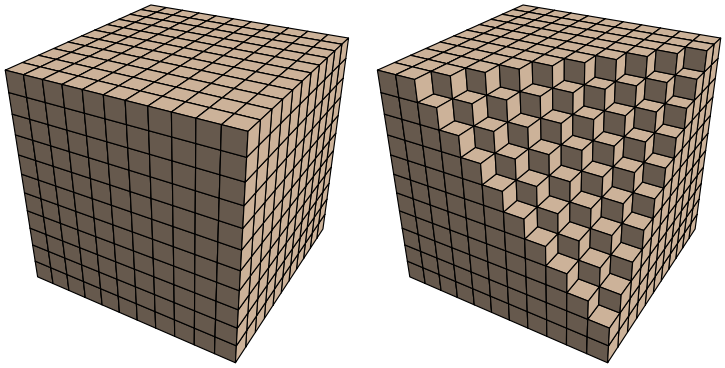
(d) Smoothed by Cosine Kernel

Figure 2.1.1: Gaussian Function Illustration

of the signal is zero. Given this definition of the Nyquist frequency, the Nyquist-Shannon theorem states that ν_N samples must be taken for each unit distance to allow accurate reconstruction of the original signal sampled. The Nyquist-Shannon theorem is a necessary condition but not sufficient, meaning that the post-process reconstruction step is also of crucial importance to reconstruct the signal accurately. Another important concept is the band-width of a signal, defined as the difference between its maximum and minimum frequency. A signal with a finite maximum frequency in its Fourier transform is called band-limited because its frequency band-width is finite. However, real signals are usually not band-limited: sharp features formed by different materials in the spatial domain correspond to step functions of infinite extent in the frequency domain, which cannot be accurately sampled and reconstructed theoretically. Even for band-limited signals, the actual maximum frequency is generally unknown. To address these uncertainties, in a sampling step a low-pass filter can be applied to restrict the band-width to a controlled value. This approach also has the advantage of being able to control the size of the volume data generated, which directly affects storage ability and rendering speed. The sampling frequency determines where the samples are placed in the domain of the function, with such placement eventually resulting in a subdivision of the domain. A mathematical function in this sense does not have an explicit analytical representation; instead it is defined by the mapping given by the sample positions and sample values. Figure 2.1.1a shows a 1D Gaussian density function between $[-2, 2]$, and Figure 2.1.1b illustrates one of its sampled representations.

As practical computing environments only possess finite resources, only a finite number of samples of the mathematical function of interest can be used, a situation having two implications. First, the function can only be described within a bounded space as exemplified in Figure 2.1.1a where only the red portion is considered visible. Second, within the bounded space, only a finite number of positions can be used to evaluate the function accurately (Figure 2.1.1c), leaving positions not sampled to a reconstruction process. The first choice is typically based on the function and the application. For example, in a CT scan of a human organ, a natural choice of the bounded space is the bounding box of the organ under concern. The second choice is directed most often by the Nyquist frequency, which demands that the sampling frequency must be at least twice the maximum frequency of the finest details in the signal to allow faithful reconstruction in a post processing algorithm. The frequency demand implies that the resolution of the spatial subdivision must be fine enough to capture the finest details in the data or that the data must be presmoothed to match the

target sampling rate. For instance, the example in Figure 2.1.1c problematically introduces high frequency on the boundary of the sampling window, while Figure 2.1.1d shows the same function modulated by a cosine function, thus producing a smooth transition on the boundary. Moreover, the Nyquist-Shannon theorem applies only to a uniform spatial subdivision, as illustrated in Figure 2.1.2 for a 3D box domain. While other subdivision schemes [77] may have a better fit depending on the applications, sampling on a uniform spatial grid is the most common approach to sampling volume data.



(a) External View (b) Cut Through View

Figure 2.1.2: Uniform Subdivision

Based on these concepts, for the research reported here

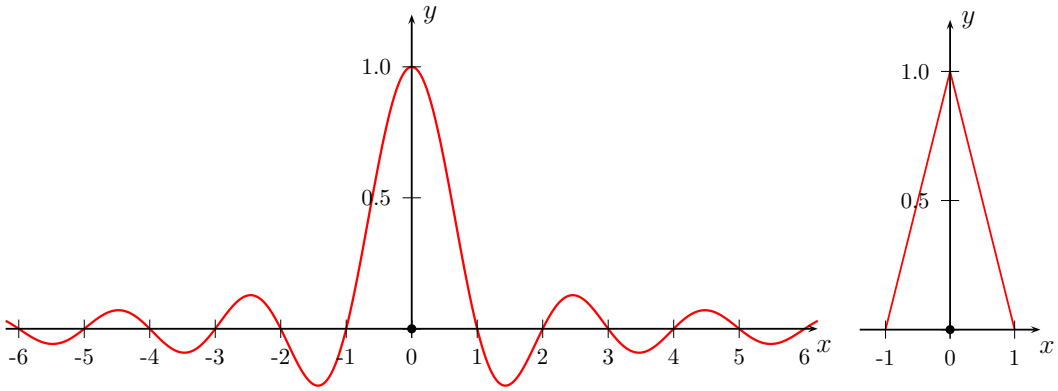
- A *signal* is the mathematical function needing to be described.
- A *domain* is the chosen bounded space that is a portion of the support of the signal.
- A *grid* is the chosen uniform subdivision of the domain.
- A *volume data* is the grid together with all the sampled values at the grid points.
- A *voxel* is a finest subdivision of the grid of volume data.

The values the signal represents are not specified to allow greater freedom for volume data as will be explained later. As briefly mentioned previously, the disadvantage of storing only a finite number of sample values of a general mathematical function is to be addressed through a reconstruction step, which creates the illusion of having the original signal available for all spatial positions. Assuming sampling above the Nyquist frequency, the theory states that a signal can be exactly recovered from

the samples through convolution with the sinc function, which is defined as

$$\text{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & \text{if} \quad x \neq 0 \\ 1 & \text{if} \quad x = 0 \end{cases} . \tag{2.1.1}$$

As shown in Figure 2.1.3a, the sinc function has infinite support. Although in almost all practical



(a) sinc function (b) tent

Figure 2.1.3: Reconstruction Kernel

volume rendering applications, the sampled signal can be safely assumed to be zero outside the domain, the definition of the sinc function still requires a convolution involving all sample values, a time-consuming operation. However, the magnitude of the sinc function converges to zero towards infinity, meaning that the contributions of the sample values decrease as their sample positions move farther from the current one. As a result, practical reconstruction algorithms use a convolution kernel of finite support to approximate the sinc function locally. The most commonly used kernel is the tent function illustrated in Figure 2.1.3b, which results in a linear interpolation of surrounding samples. In 3D, the linear interpolation expands to trilinear interpolation, defined as

$$\begin{aligned} f(\vec{x}) &= (1 \quad -t_z) f(\vec{x}_{**0}) + t_z f(\vec{x}_{**1}) \\ &= (1 \quad -t_z) \left[(1-t_y) f(\vec{x}_{*00}) + t_y f(\vec{x}_{*10})\right] \\ &\quad\quad +t_z \left[(1-t_y) f(\vec{x}_{*01}) + t_y f(\vec{x}_{*11})\right] \\ &= (1 \quad -t_z) \left\{(1-t_y) \left[(1-t_x) f(\vec{x}_{000}) + t_x f(\vec{x}_{100})\right] + t_y \left[(1-t_x) f(\vec{x}_{010}) + t_x f(\vec{x}_{110})\right]\right\} \\ &\quad\quad +t_z \left\{(1-t_y) \left[(1-t_x) f(\vec{x}_{001}) + t_x f(\vec{x}_{101})\right] + t_y \left[(1-t_x) f(\vec{x}_{011}) + t_x f(\vec{x}_{111})\right]\right\} \end{aligned}$$

and illustrated in Figure 2.1.4.

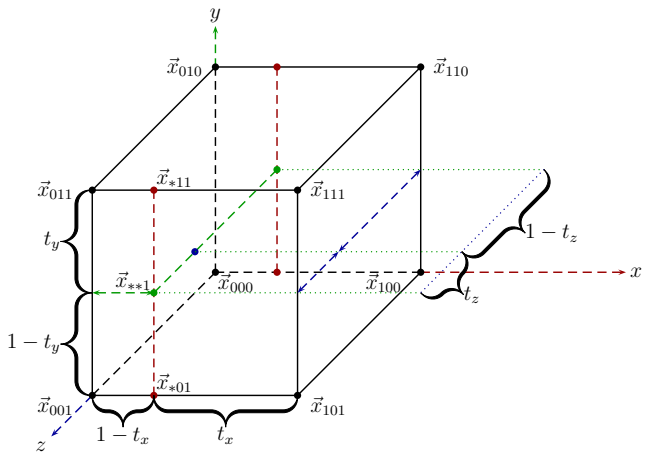


Figure 2.1.4: Trilinear Interpolation

2.2 Volume Visualization

The problem of the visualization of volume data, such as in CT scans, existed long before volume rendering came into maturity. As a result, the early techniques of volume visualization were significantly influenced by surface-based rendering techniques. A large body of work focuses on interpreting volume data through the extraction or construction of polygonal surfaces, meaning that the visualization of the volume data is then the result of the visualization of polygonal meshes, for which many rendering techniques are available. One of the seminal methods that falls into this category is the Marching Cubes algorithm [59] and its related methods [13, 21, 47, 58, 65, 66].

2.2.1 Marching Cubes

The Marching Cubes algorithm, a triangle mesh extraction algorithm, is based on scalar valued volume data; i.e., the dependent variable of the signal is a scalar. In addition, it requires a special scalar value v called the isovalue. The volume data and the isovalue together define an isosurface that can be extracted using the Marching Cubes algorithm. In the mathematical sense, the isosurface is the contour formed by the signal, i.e. the volume data, with respect to the isovalue. The isovalue must fall within the range of the signal; otherwise, no surface can be found. The algorithm processes all edges of the underlying grid of the volume data by looking for the intersections of the signal with the isovalue, i.e. adjacent samples falling above and below the isovalue. It evaluates the signal on any point along an edge by linearly interpolating the values stored at its two end points as illustrated in Figure 2.2.1.

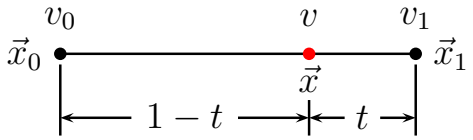


Figure 2.2.1: Marching Cubes Interpolation Scheme

If an intersection is found, i.e. there exists $t \in [0, 1]$ such that the isovalue $v = (1-t)v_0 + tv_1$ where v_0 and v_1 are the scalar values stored at the two end points of the edge under concern, then a vertex of the extracted mesh is generated by computing $\vec{x} = (1-t)\vec{x}_0 + t\vec{x}_1$ where $\vec{x}_0$ and $\vec{x}_1$ are the spatial positions of the corresponding grid points implied by the topology of the grid. Figure 2.2.1 illustrates this process. The resulting vertices give a complete definition of the geometry of the

isosurface.

To fully construct the isosurface as a polygonal mesh, however, Marching Cubes also needs to generate its topological connection, accomplished by considering all the voxels of the volume data. For each one, it forms a configuration index by comparing the isovalue with the scalar values stored at the corner grid points of the voxel. The result gives a total of 256 possibilities corresponding to the 256 pre-identified intersection topologies of the isosurface with the voxel under concern. By identifying the topology for each voxel, Marching Cubes generates the topology of the isosurface. The 256 pre-identified topological configurations, however, can be reduced by symmetry to only 15 distinct cases, as illustrated in Figure 2.2.2, thus simplifying the lookup table usually used when implementing the Marching Cubes algorithm.

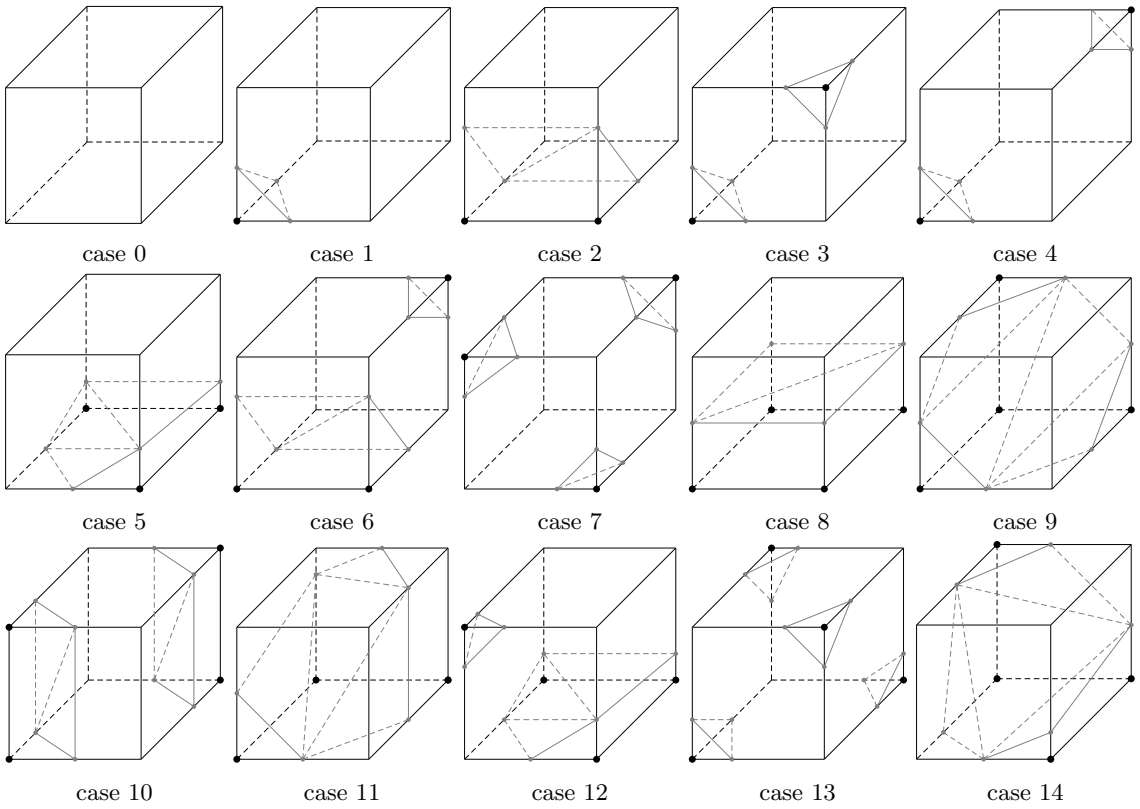


Figure 2.2.2: Marching Cubes 15 Case Table (Redrawn from Lorensen and Cline [59])

However, as discussed in [12, 64, 65, 66] among others, the original Marching Cubes algorithm has a potential defect in that it can generate inconsistent topologies for adjacent voxels, effectively creating holes in the extracted mesh. It can also generate skinny triangles when the

isovalue is close to a stored value. Moreover, for large volume datasets, Marching Cubes generates a very large number of triangles, which can greatly slow the generation and rendering process. Related research has improved Marching Cubes [19, 21, 27, 28, 40, 41, 46, 75, 78, 79, 87]. However, these studies all rely on surface-based rendering that does not utilize the volume data to its full potential as surface-based graphics can visualize only a few contours of the volume data. The fundamental problem is that surface-based graphics concerns light interactions only on the surface of shapes. As intuitive as it is, this approach has two important restrictions:

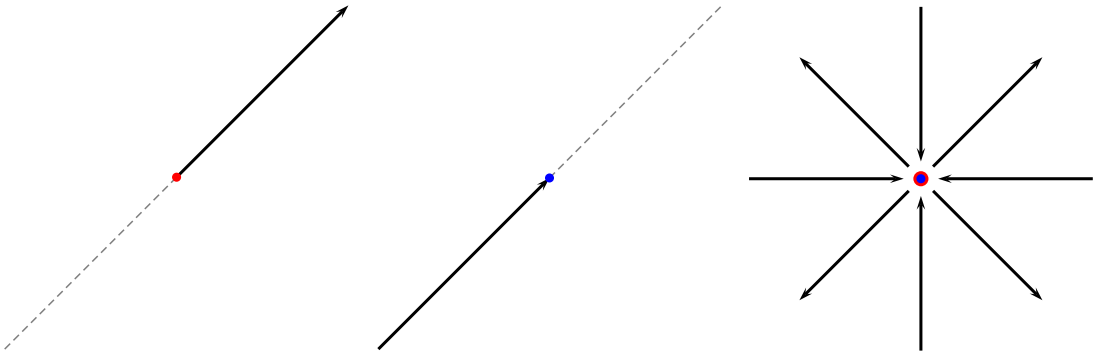
- It requires all renderable shapes to have well-defined boundaries.
- It considers light interactions only on the boundaries while ignoring them over the volumes.

As a result, many interesting lighting effects cannot be modeled; specifically, anything that is not completely opaque or fairly regular cannot be properly rendered or easily handled, for example wax, fog and hair. The advantage of volumetric methods is that they are able to capture the distribution of irregular and complex quantities over a 3D domain involving numerous visualization targets and multiple interactions. One of the most effective methods for capturing lighting effects in volume data is volume rendering, which has been the focus of much research over the past three decades.

2.2.2 Volume Rendering

Much of the discussion here about volume rendering and its related concepts is summarized from Engel *et al.* [23], which can be referred to for additional details. Volume rendering is based on geometric optics as in the case of surface-based graphics. Geometric optics considers that light propagates along a straight line unless interactions with participating medium change the direction. The fundamental difference is that volume rendering considers such interactions over a continuous spatial domain rather than only at object surfaces or boundaries. In volume rendering, three types of light-medium interactions are considered: emission, absorption and scattering, as illustrated in Figure 2.2.3.

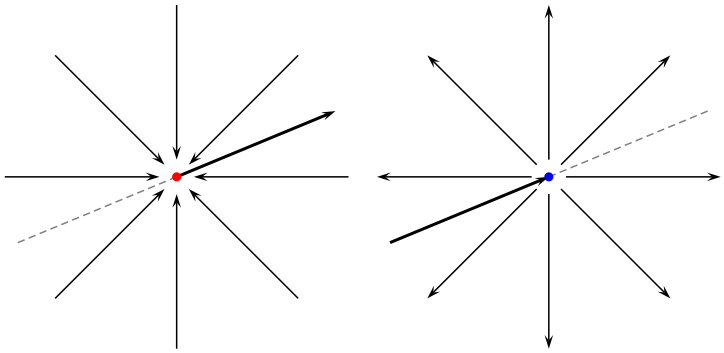
Emission (Figure 2.2.3a) is the ability of the material to emit light on its own, while absorption (Figure 2.2.3b) describes how light energy is consumed. Scattering (Figure 2.2.3c) is the ability of the material to affect the direction of light propagation. Depending on whether the light wavelength is changed, there are two types of scattering, inelastic and elastic, the type important



(a) Emission (b) Absorption (c) Scattering

Figure 2.2.3: Types of Light Medium Interactions

to the research here. Elastic scattering does not affect light wavelength, meaning it does not affect light energy. For interactive rendering systems, only elastic scattering is considered.



(a) In-scattering (b) Out-scattering

Figure 2.2.4: Types of Scattering

Evaluating all three types of interactions precisely and accurately can be expensive. As a result, several simplified models have been proposed, including absorption only, emission only, absorption and emission, single scattering, shadowing, and multiple shadowing. Of particular interest here is the single scattering model, which includes the absorption and emission effects as well as first order scattering effects. In volume rendering, only light energy along the view direction is of interest. As a result, only direct scattering of light rays from light sources into the view direction is considered. This model significantly lowers the computational complexity of a multiple scattering model, while still offering local illumination, thus maintaining a good balance between efficiency and realism. Since only light energy scattered into the view direction is received by the virtual camera, the scattering effect contains two subclasses, in-scattering and out-scattering as illustrated in Figure 2.2.4.

Since volume rendering considers light-medium interaction over a continuous domain, these interactions are mathematically captured by an integral form that is known as the volume rendering integral. When considered in the view direction, parameterized by its arc length s, this integral form is simplified to

$$I(D) = I(s_0)e^{-\int_{s_0}^{D} \kappa(t)\,\mathrm{d}t} + \int_{s_0}^{D} q(s)e^{-\int_{s}^{D} \kappa(t)\,\mathrm{d}t}\,\mathrm{d}s \tag{2.2.1}$$

where $I(s_0)$ is the light radiance entering the volume from the background at position $s = s_0$ and $I(D)$ is the radiance leaving the volume at position $s = D$. The first term in Equation 2.2.1 describes light intensity from the background attenuated by the volume and the second one the integral contribution of the light energy increase attenuated by the participating medium for the remaining distances to the position at $s = D$. At any position s, the energy increase $q(s)$ accounts for any emissive energy from the material and the in-scattering effect from any light source.

In practice, Equation 2.2.1 is numerically evaluated by a Riemann sum with the attenuation effect computed as a geometric series. Specifically,

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} (1 - \alpha_i) \tag{2.2.2}$$

where $\alpha_i = 1 - T_i = 1 - T(s_{i-1}, s_i)$ is the opacity of the medium and $c_i = \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)\,\mathrm{d}s$ is the energy influx integral for the interval $[s_{i-1}, s_i]$. The transparency of the medium between $[s_1, s_2]$ is defined as

$$T(s_1, s_2) = e^{-\tau(s_1,s_2)} = e^{-\int_{s_1}^{s_2} \kappa(t)\,\mathrm{d}t} \tag{2.2.3}$$

where $\kappa(s)$ is the absorption coefficient at position s and describes the ability of the medium to consume light energy, and $\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(t)\,\mathrm{d}t$ is the *optical depth* that gives the expected amount of light radiance decrease over $[s_1, s_2]$.

Equation 2.2.2 effectively evaluates light attenuation and color accumulation effects along a given direction. Depending on the starting point of the iteration, there are two types of iteration order, front-to-back composition and back-to-front composition. Front-to-back composition begins evaluating the integral from the point that a light ray enters the volume domain from a viewpoint, advancing back into the volume. It is opposite to the natural order of light ray traversal towards the viewpoint, but it has the advantage of supporting early ray termination [57], which is an important

acceleration technique. The front-to-back composition is carried out in a ray marching scheme, where the update is given by

$$\begin{array}{rcccl} C_{n+1} & \leftarrow & C_n & + & (1-A_n) \quad c, \\ A_{n+1} & \leftarrow & A_n & + & (1-A_n) \quad \alpha, \end{array} \tag{2.2.4}$$

where α and c describe the opacity and color at the current sample position and A_n and C_n are the accumulated opacity and color at step n. Determining α and c in a physically realistic and accurate manner is often not possible and is not always necessary. They are usually chosen based on artistic and perceptual needs. Typically, the mapping from the volume data to opacities and colors is artificially given, and is called the transfer function $f : R \rightarrow [0, 1]$ for opacity or $f : R \rightarrow [0, 1]^3$ for RGB color.

The single scattering model can include the local illumination effect, which requires a normal direction for each point where local illumination needs to be computed. By definition, a normal is perpendicular to a surface at any point. For volume data, the gradient vector at position $\vec{x}$ is perpendicular to an imaginary isosurface passing through $\vec{x}$. The gradient vector for a function $f : R^3 \rightarrow R$ at position $\vec{x}$ is given by

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f(\vec{x})}{\partial x} \\ \frac{\partial f(\vec{x})}{\partial y} \\ \frac{\partial f(\vec{x})}{\partial z} \end{bmatrix}, \tag{2.2.5}$$

and the normal at $\vec{x}$ is defined as

$$\hat{n}(\vec{x}) = \frac{\nabla f(\vec{x})}{\|\nabla f(\vec{x})\|}, \quad \text{if } \|\nabla f(\vec{x})\| \neq 0. \tag{2.2.6}$$

The gradient vectors for points in homogeneous regions are of zero magnitude, in which case the normal vectors are not well defined. This fact implies that the local illumination model stops working at these points, a direct consequence of the model assumption that local illumination only occurs at material boundaries. However, having no normal does not imply no light-medium interaction can happen in homogeneous regions. In fact, light-medium interactions still occur as usual, meaning that light intensity will be diminished when a light ray goes through homogeneous regions. Therefore, further light interaction will be based on the decreased light intensity, which should cause a shadow

effect. However, the single scattering model does not take this shadow effect into consideration, a fact that is reflected in its assumption that all light rays reach all points without being impeded.

In general, the analytical form of the signal defined by a volume dataset is not known. It follows that it is impossible to find the explicit form for the gradient vector as well. Therefore, numerical techniques are relied on to solve for the gradient vector at any point. The most common technique is to use finite difference to estimate the gradient vector. This method, which is fast and easy to implement, gives reasonable accuracy. There are three finite difference schemes that can be used to estimate the gradient vector: the forward difference, the backward difference and the central difference, defined respectively by

$$\begin{aligned} f'(x) &= \tfrac{f(x+h)-f(x)}{h} + o(h), \\ f'(x) &= \tfrac{f(x)-f(x-h)}{h} + o(h), \\ f'(x) &= \tfrac{f(x+h/2)-f(x-h/2)}{h} + o(h^2), \end{aligned} \tag{2.2.7}$$

where h is the step size between samples. Central difference is favored in most cases because of its increased accuracy. Therefore, the gradient vector at position (x, y, z) is numerically evaluated as

$$\nabla f(x,y,z) \approx \frac{1}{2h} \begin{bmatrix} f(x+h,y,z) - f(x-h,y,z) \\ f(x,y+h,z) - f(x,y-h,z) \\ f(x,y,z+h) - f(x,y,z-h) \end{bmatrix}. \tag{2.2.8}$$

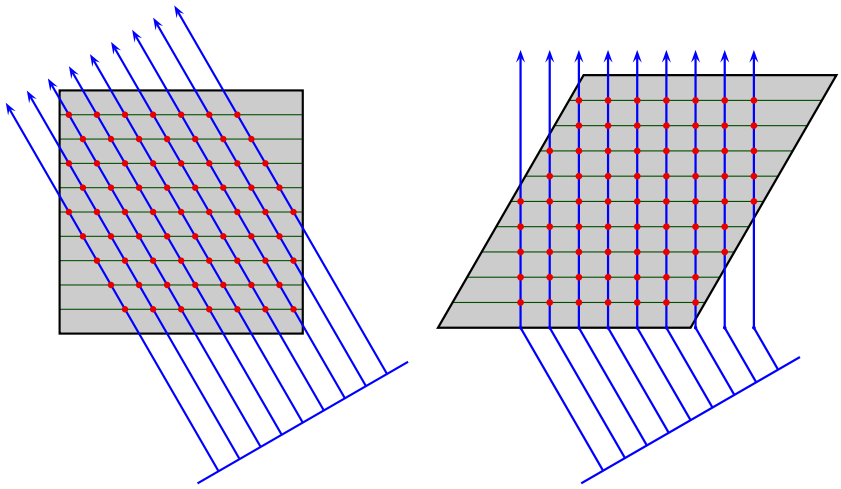
The difference step size h in this equation is a complete voxel size instead of a half voxel size as in Equation 2.2.7 to avoid evaluating the signal at non-grid points, a process that requires interpolation.

2.2.3 General Pipeline

The general volume rendering pipeline works as follows. For each ray from the viewpoint, sample positions within the volume are determined. For each sample position, a value, which is obtained from the volume data through trilinear interpolation, is mapped to opacity and color through the transfer function. The gradient vector is also computed for each sample position using central difference, which, if non-zero, is then used to compute the local illumination. The opacity and color are composited according to Equation 2.2.4.

Several different volume rendering algorithms have been explored for both CPU and GPU

implementations. The most relevant algorithms for the problem investigated here are the shear-warp method, the texture slicing method and the ray marching algorithm.



(a) Original Traversal Order (b) Sheared Volume Traversal Order

Figure 2.2.5: Shear Warp Method

The shear-warp algorithm [52] is an early technique based on graphics cards. Figure 2.2.5a shows the sample positions in an orthographic projection rendering. To generate the same set of sample positions, the shear-warp algorithm shears the volume using a set of planes that are aligned with the axis directions of the volume as shown in Figure 2.2.5. The shear operation is necessary because early graphics hardware supported only 2D textures, with a 3D texture being stored as a stack of 2D textures. Therefore, arbitrary intersection between a plane and the 3D volume is not supported. Perspective projection is possible by applying a corresponding scale transformation of the 2D proxy planes.

Because of further advancement in GPU architectures, 3D texture support is now available. As a result, a volume dataset can be stored in device memory directly. Corresponding sampling ability has also been added to allow a 2D plane to intersect the 3D volume in any way specified. This ability gives greater freedom in placing a 2D proxy plane for the purpose of compositing. One natural consequence is that the texture slicing method (Figure 2.2.6) always places planes perpendicular to the view direction and iterates them in either front-to-back or back-to-front order to give the desired compositing scheme. Compared with the shear warp algorithm, this method does not require shearing the volume, making it more intuitive to work with.

Shear warp and texture slicing are more or less affected by hardware architecture or data

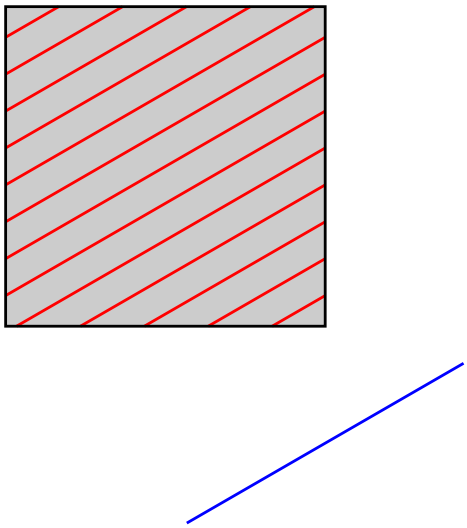


Figure 2.2.6: Texture Slicing Method

structure restrictions. Working with a dense volume data structure, uniform grid topology and modern graphics hardware, however, gives the freedom to use a more intuitive and powerful method known as ray tracing. In the ray tracing algorithm, the virtual camera emits rays through the screen and into the environment, accumulating the light medium interactions along the path of each ray. The virtual screen is discretized into pixels. A ray is shot through each pixel, and the light-medium interactions of the ray with the volume are then computed according to the front-to-back compositing scheme based on Equation 2.2.4. The benefit of ray tracing is a very simple and intuitive algorithm that traces light propagation in the reverse direction, following paths that almost exactly match those of light rays in nature. Modern graphics hardware architectures have direct support for 3D texture memory, fast trilinear interpolation, cheap multithreading functionality and vector arithmetic. Thus, they are very friendly to ray tracing implementation. Traditionally, ray tracing also includes tracing reflection and refraction rays. Both of them, however, are usually too expensive to be included in an interactive volume rendering process.

In practice, efficient volume rendering algorithms trace only the primary ray. This simplified version of ray tracing for volume rendering is known as ray marching, so named to describe its principle of marching a sample position along the traced ray direction through the volume to accumulate light-medium interaction effects. Figure 2.2.7 shows the traversal order in a 2D context, where the red dots are the sample positions. This algorithm is the one used in our implementation.

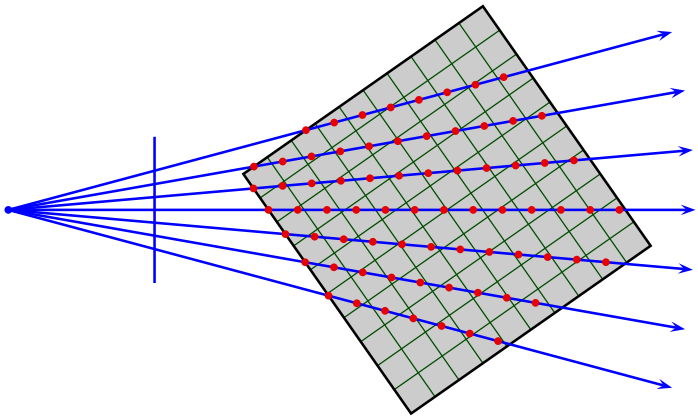


Figure 2.2.7: Ray Marching Traversal Order

Even with modern graphics hardware, volume rendering can be inefficient, as ray marching is a slow process, especially for large datasets or images. There has been much research focused on acceleration techniques, including early ray termination [57], empty space skipping and space subdivision methods [14, 16, 53, 85], adaptive resolution representation [43], pre-integrated volume rendering [24] and virtual sampling [55], among others. Virtual sampling uses cubic interpolation to compute additional samples between two regular samples during ray marching. Such interpolation requires information from four adjacent regular samples. The two outer samples are required by the cubic interpolation kernel. Figure 2.2.8 illustrates the principle of virtual sampling. Here, the blue

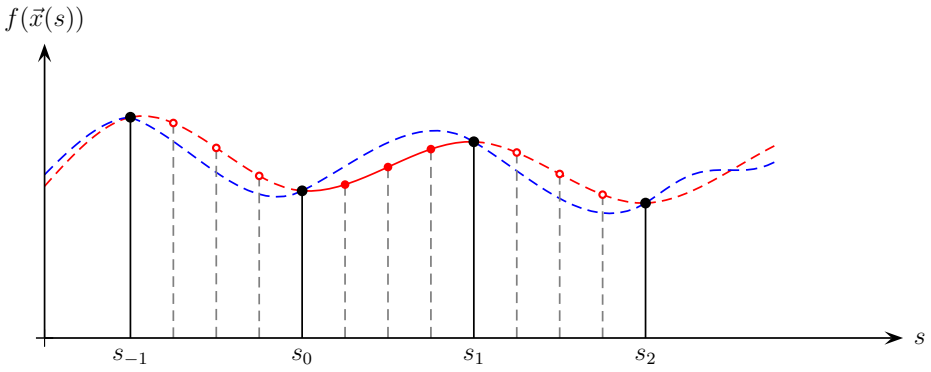


Figure 2.2.8: Virtual Sampling

dashed curve is the signal to be reconstructed. The dashed and solid red curve is reconstructed by virtual sampling. The black dots are regular samples in ray marching. Virtual sampling gives extra virtual samples between each adjacent pair of regular samples, marked as solid or hollow red dots in Figure 2.2.8. The difference between hollow and solid dots is that the generation of solid dots

requires information from all four black dots, not only the middle two containing the solid segment, a direct consequence of the cubic continuity assumption made by virtual sampling. The hollow dots, therefore, require information from two more regular samples beyond the current scope, samples not drawn in Figure 2.2.8. The benefit of additional virtual samples is that they simulate a sampling scheme whose ray marching step size is one fourth of the current step size, effectively simulating a finer ray marching than the current one. The benefit of a finer step size is higher image quality and the benefit of virtual sampling is to give the same image quality with approximately a 2 to 3 fold increase in speed compared to regular ray marching done at the virtual step size.

In summary, this section introduced the idea of volume data and its applications. In addition, it described volume rendering, a technique for visualizing volume data. These two combined give the ability to describe optical phenomena that cannot be captured by surface-based visualization methods. Then implementation concerns and techniques, along with their advantages and disadvantages, were discussed. Finally, volume ray marching and its acceleration techniques, the primary algorithm used in this work, were discussed. As briefly introduced in Chapter 1, the primary goal of this research is to improve human perception of shape in a volume rendering environment in the hope that such a technique could see a wide range of volume rendering applications. Therefore, volume rendering is not an end for this research, nor is it the only need. Existing research shows that certain types of line elements covering a shape can help humans better understand it. Production of these lines is a key element of this research, the topic discussed next.

2.3 Descriptive Line Elements

While for any shape there are an infinite number of ways of drawing a line on its surface, not every one assumes the same importance in describing it. Intuitively, lines with special mathematical properties may play a significant role in helping humans comprehend shape. In addition, other well-designed line patterns have also been found to achieve a similar effect. Since the early days of computer graphics, people have explored many ways to algorithmically construct lines that are effective in conveying surface shape [10, 15, 20, 22, 50, 74]. These techniques have promising applications in fields where this understanding is the primary perception goal, such as in textbooks for explanation purposes, in illustrative drawings and sketches, or in visualization of scientific or medical data.

The primary difficulty of finding effective lines for abstraction is that the human visual perceptual mechanisms are not fully understood. As a result, it is unclear what types of lines are most effective. However, scientists have drawn conclusions by studying how artists choose to draw lines [15]. One important mathematical tool for such studies has been differential geometry which is concerned with problems of defining differential properties on non-planar geometrical surfaces. While not all shapes in nature are differentiable, a large number of interesting shapes are and, for those that are not, a portion can have differential properties. Therefore, differential geometry has a broad range of applications, including guiding computers to identify important shape-telling lines. Based on their mathematical definitions, these lines have been classified in the literature into the following categories:

Elevation lines – Intersections between geometry and a set of planes parallel to the ground.

Cutting lines – Intersections formed by a set of planes perpendicular to the view direction.

Silhouettes – Boundaries between objects and their backgrounds.

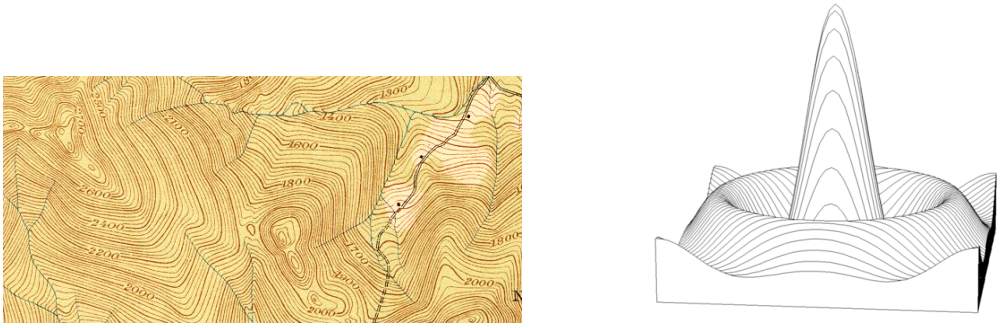
Occluding contours – Lines where depth continuity breaks.

Isophotes – Contours on which light intensity is constant.

Suggestive contours – Lines where radial curvature reaches local minima.

Ridges and valleys – Lines on which curvature reaches local extrema.

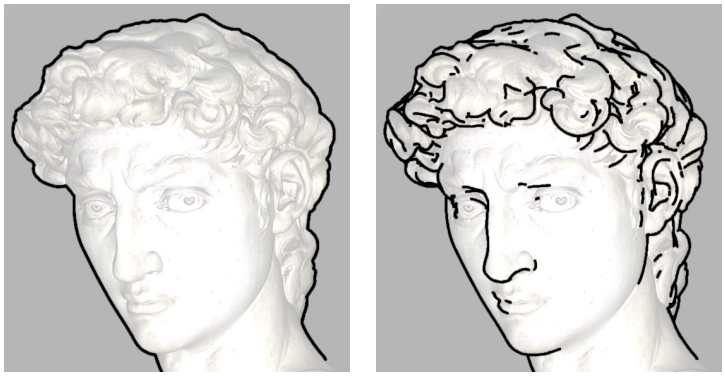
Elevation lines (Figure 2.3.1a) and cutting lines (Figure 2.3.1b) are the most straightforward form of line elements on geometric objects. The former are determined without reference to the view direction while the latter depend on the view direction. In the language of differential geometry, these lines are zero-order features of the geometry because their formation relies only on the definition of the surface without utilizing any of its differential properties. As a result, they have limited connection to the geometry they describe, and, hence limited ability to express its shape.



(a) Elevation Lines (USGS) (b) Cutting Lines (Rusinkiewicz *et al.* [73])

Figure 2.3.1: Zero Order Lines

The first-order lines include occluding contours as well as silhouettes, and isophotes. Figure 2.3.2 illustrates silhouettes and occluding contours of the same geometrical model. They involve



(a) Silhouette (b) Occluding Contours

Figure 2.3.2: First Order Lines (Rusinkiewicz *et al.* [73])

a first-order differential property of the geometry, i.e., the normal of the surface which is parallel to the gradient vector. The gradient vector is found by determining the first-order derivative of the surface in all axis directions individually.

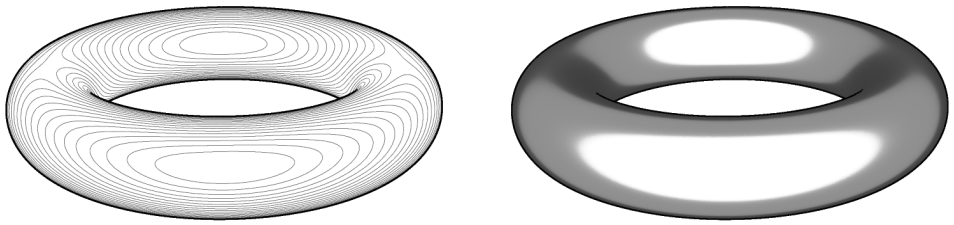
Silhouettes and occluding contours share the same mathematical definition. Intuitively,

occluding contours form the boundary between a foreground object and a background one. These two objects may be two different parts of the same geometry viewed from angles such that one part occludes the other. Another interpretation is that occluding contours are places where depth discontinuity occurs. For smooth surfaces, occluding contours are mathematically defined as the set of points $\vec{x}$ on a surface such that

$$\langle \hat{n}(\vec{x}), \vec{c} - \vec{x} \rangle = 0 \tag{2.3.1}$$

where $\hat{n}(\vec{x})$ denotes the normal at $\vec{x}$ and $\vec{c}$ is the virtual camera position or the current viewpoint. This equation can be interpreted in a second way: occluding contours are the places where the geometry begins to turn back to the viewer, thus forming boundaries on the geometry between visible and invisible regions. It is in this sense that silhouettes can be considered as a special case of occluding contours, with the additional property that silhouettes also define the boundary on the view plane of the projected geometry. Equation 2.3.1 shows the view dependent nature of silhouettes and occluding contours, a situation that has advantages as well as disadvantages. On the one hand, these lines are more likely to convey shapes rather than merely some types of special marks on the geometry. On the other hand, it means that these lines need to be recomputed for each frame. It is also more difficult to use them in a stereoscopic display environment, where images must be computed from two viewpoints.

Isophotes, on the other hand, can be view-independent. For example, the lines in Figure 2.3.3a do not require the existence of a viewer. Nonetheless, isophotes do require a reference



(a) Isophotes (b) Toon Shading Boundary

Figure 2.3.3: Isophotes (Rusinkiewicz *et al.* [73])

point, which, unlike for occluding contours, does not have to be the view point. Thus, this reference point is more arbitrary as well as less obvious. Assuming such a point $\vec{r}$, isophotes are the set of points $\vec{x}$ where the surface normal forms a constant angle with respect to the vector from $\vec{x}$ to $\vec{r}$.

Thus, isophotes are points $\vec{x}$ such that

$$\langle \hat{n}(\vec{x}), \vec{r} - \vec{x} \rangle = v. \tag{2.3.2}$$

Since $\langle \hat{n}(\vec{x}), \vec{r} - \vec{x} \rangle$ forms a continuous scalar function on the surface, Equation 2.3.2 can be thought of as defining the contour of the scalar function with respect to an isovalue v. In this sense, Equation 2.3.2 includes Equation 2.3.1 as a special case, where the reference point is the view position and the isovalue is fixed at zero. This extra freedom gives the possibility of other useful applications, one example being choosing a light position as the reference point $\vec{r}$ in Equation 2.3.2 and a few constant values for v. This results in a family of non-intersecting lines, each having a constant light intensity. As shading significantly enhances shape perception, isophotes hold a similar property. In addition, the boundaries between toon shading [54] regions are also isophotes as illustrated in Figure 2.3.3b.

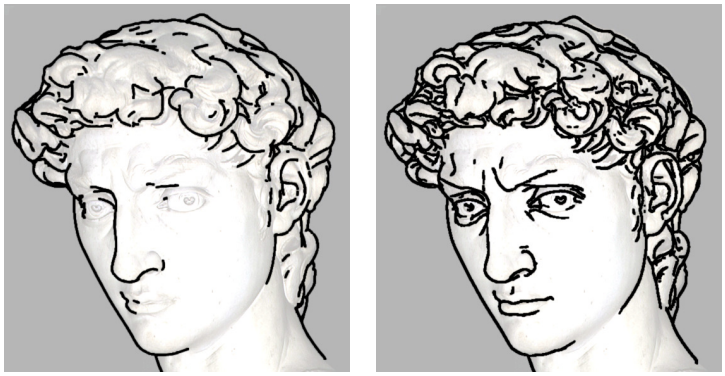
First-order differential properties only provide as much information as exemplified in occluding contours and isophotes. To obtain more interesting shape-conveying line elements, higher order differential attributes have to be considered. Before discussing line elements with higher order differential properties, we introduce curvature and its related concepts as they are second order differential attributes and form the foundation for the following discussions.

The curvature $\kappa(\vec{p})$ at a point $\vec{p}$ on a plane curve $\mathcal{C}$ is a scalar defined as $1/R$ where R is the radius of the osculating circle, which is the circle that passes through the point $\vec{p}$ and is tangent to the curve at $\vec{p}$. Since the osculating circle shares the same tangent vector and curvature at $\vec{p}$ as $\mathcal{C}$, it is the best circle locally approximating $\mathcal{C}$ at $\vec{p}$. The curvature measures how much the curve bends at $\vec{p}$. Intuitively, when R is large, the circle best fitting the curve has a large radius, thus implying that the local section of the circle at $\vec{p}$ appears flat, or bends relatively weakly. Likewise, $1/R$ assumes a relatively small value. Therefore, a small curvature at $\vec{p}$ indicates a weak tendency for the curve to bend at $\vec{p}$; while a large curvature describes a strong tendency to bend. At an extreme situation, the curvature for every point on a straight line is 0 as the osculating circle has an infinite radius. A circle of radius r is another illustrating case since it has a constant curvature $1/r$.

The curvature at a point $\vec{p}$ on a surface $\mathcal{S}$ is based on the curvature of plane curves. In particular, let $\hat{n}(\vec{p})$ be the unit vector normal to the surface at $\vec{p}$ and $\hat{t}(\vec{p})$ be a unit vector in the plane $\mathcal{P}_{\hat{t}}$ tangent to $\mathcal{S}$ at $\vec{p}$. By definition, $\hat{n}(\vec{p})$ is perpendicular to $\hat{t}(\vec{p})$, thus defining a plane $\mathcal{P}$ containing $\vec{p}$, $\hat{n}(\vec{p})$ and $\hat{t}(\vec{p})$, and intersecting with $\mathcal{S}$. The intersection $\mathcal{P} \cap \mathcal{S}$ is necessarily contained

by both $\mathcal{P}$ and $\mathcal{S}$. Therefore, $\mathcal{P} \cap \mathcal{S}$ is a plane curve associated with the tangent direction $\hat{t}(\vec{p})$ and the curvature $\kappa_{\hat{t}}(\vec{p})$ of $\mathcal{P} \cap \mathcal{S}$ at $\vec{p}$ is well defined. As there are an infinite number of unit directions in the tangent plane of $\mathcal{S}$ at $\vec{p}$, there are an infinite number of such curvatures $\kappa_{\hat{t}}(\vec{p})$. The function $\kappa_{\hat{t}}(\vec{p}) = \kappa(\hat{t}; \vec{p}) : R^3 \mapsto R$ thus maps every tangent direction $\hat{t}$ to a scalar known as the normal curvature at $\vec{p}$ in that direction. Assuming $\mathcal{S}$ is sufficiently smooth locally at $\vec{p}$, the set $\mathbb{K}(\vec{p}) = \{\kappa_{\hat{t}}(\vec{p}) : \hat{t} \in \mathcal{P}\}$ is infinite and bounded, and the supremum and infimum of $\mathbb{K}(\vec{p})$ are known as the principal curvatures of $\mathcal{S}$ at $\vec{p}$, denoted as κ_1, κ_2 respectively. As long as $\kappa_1 \neq \kappa_2$, there are precisely two orthogonal tangent directions $\hat{t}_1$ and $\hat{t}_2$ along which the normal curvature realizes the principal curvatures. These tangent directions are known as the principal curvature directions, corresponding to κ_1 and κ_2 respectively. When $\kappa_1 = \kappa_2$, the principal curvature directions are not unique, as is the case everywhere on a sphere. The mean curvature at $\vec{p}$ is defined as $H = \frac{\kappa_1 + \kappa_2}{2}$ and the Gaussian curvature is $K = \kappa_1 \kappa_2$.

Closely related to the concept of curvature is a family of lines known as the suggestive contours [18]. They are not true contours; instead they are "almost" contours in the current view, complementing true contours as illustrated in Figure 2.3.4. They possess two advantageous proper-

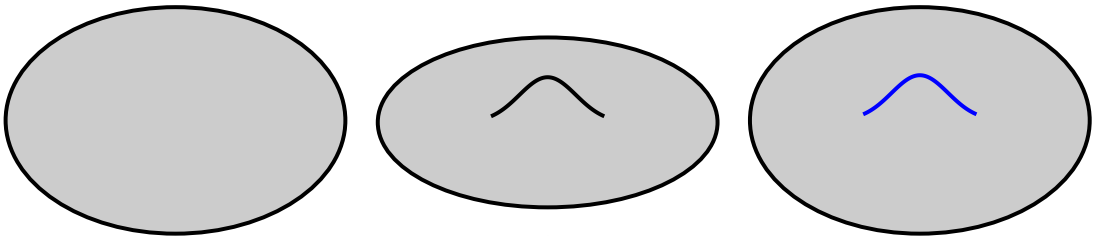


(a) Contours Only (b) Suggestive Contours

Figure 2.3.4: Second Order Lines (Rusinkiewicz *et al.* [73])

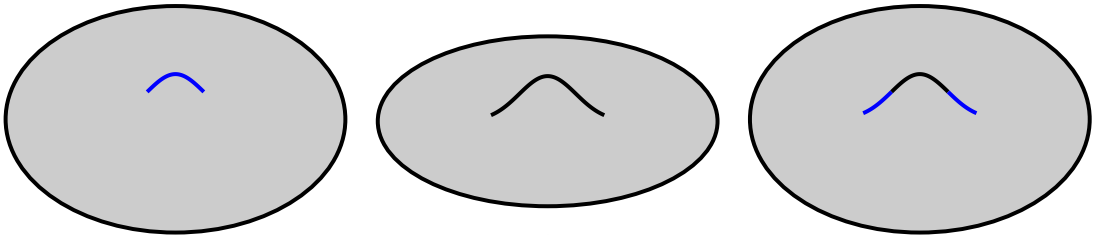
ties. First, they become true contours in nearby views. As a result, they show anticipation in terms of where the contours will emerge as the view position moves as illustrated in Figure 2.3.5. Second, they extend the contours of the current view smoothly as shown in Figure 2.3.6.

In Equation 2.3.1, the quantity $\langle \hat{n}(\vec{x}), \vec{x} - \vec{c} \rangle$ is defined for all points on a surface. Therefore, it defines a function $f : R^3 \to R$ on the surface. Suggestive contours are the zeros of the first-order derivative of f, the second-order derivative of which is positive. Moreover, the derivative of f is



(a) Original View (b) Contour in Nearby View (c) Suggestive Contour

Figure 2.3.5: Suggestive Contours Showing Anticipation (Redrawn from DeCarlo *et al.* [18])



(a) Original View (b) Contour in Nearby View (c) Suggestive Contour

Figure 2.3.6: Suggestive Contours Extending Contours (Redrawn from DeCarlo *et al.* [18])

related to curvature in that it shares the same zeros as the radial curvature, which is defined as the curvature at position $\vec{x}$ in direction $\vec{w}$, where $\vec{w}$ is the projection of the view vector $\vec{c} - \vec{x}$ onto the tangent plane at $\vec{x}$. Figure 2.3.7 illustrates the relationship between $\vec{c}$, $\vec{x}$ and $\vec{w}$. Thus, the formal

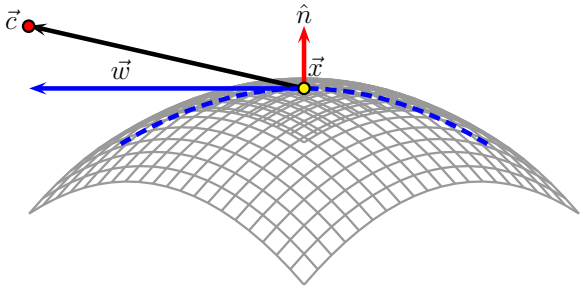


Figure 2.3.7: Radial Curvature

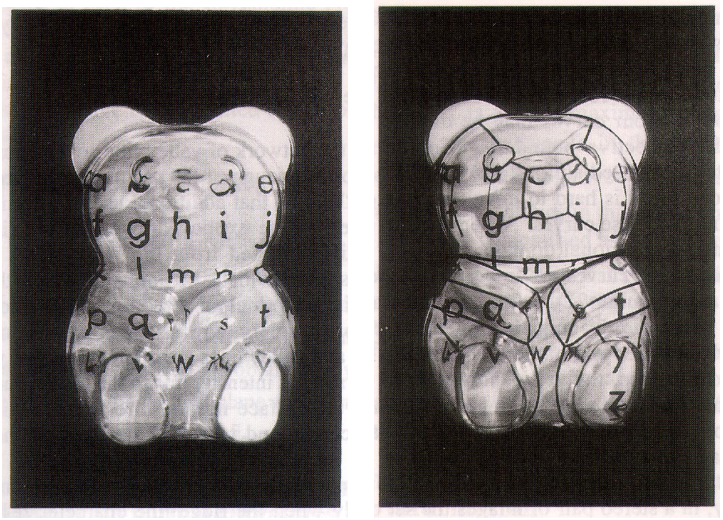
definition of suggestive contours is the set of points $\vec{x}$ on a surface such that

$$\begin{aligned} \kappa_{\hat{r}}(\vec{x}) &= 0 \quad \text{and} \\ D_{\vec{w}}(\kappa_{\hat{r}}(\vec{x})) &> 0 \end{aligned} \tag{2.3.3}$$

where $\kappa_{\hat{r}}(\vec{x})$ is the radial curvature at position $\vec{x}$ in the radial direction $\hat{r}$ and $D_{\vec{w}}(\cdot)$ forms the directional derivative in the direction $\vec{w}$ for its operand quantity. Satisfying the relationships in 2.3.3

is equivalent to finding points where the derivative of f is zero and its second-order derivative is positive. Therefore, the set of points satisfying 2.3.3 are suggestive contours. The geometrical explanation of suggestive contours is that they are points where radial curvature reaches a local minimum.

The next order of differential properties includes ridges and valleys [49], which are view independent line elements. Figure 2.3.8 shows ridges and valleys compared with a plain photo. As



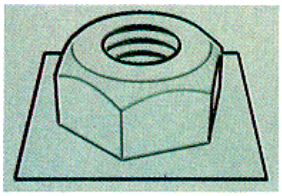
(a) Plain Picture (b) Ridges and Valleys

Figure 2.3.8: Third Order Lines (Interrante *et al.* [37])

defined by Koenderink [49], ridges are points on a surface where the normal curvature assumes a local maximum in the principal curvature direction associated with the largest positive curvature. Similarly, valleys are points where normal curvature reaches a local minimum in the principal curvature direction associated with the largest negative curvature. Given their definitions, the process for identifying a valley point $\vec{x}$ is to determine if the largest principal curvature $\kappa_1 < 0$, and if so, if $\vec{x}$ is a local minimum of normal curvature in the principal curvature direction that corresponds to κ_1. For volume data, Interrante *et al.* [37] suggested this process can be done through using a small step along the principal curvature direction $\hat{e}_1$ from point $\vec{x}$ to $\vec{x}'$ to determine if $\kappa_1(\vec{x}) \leq \kappa_1(\vec{x}')$. A similar approach can be used to determine if a point is located on a ridge.

In addition to these lines, other line elements include, for example, creases [74], lines commonly found in polyhedral objects where the joining faces form a sharp dihedral angle at their connecting edge. Figure 2.3.9a gives an example. Apparent ridges [42] are points that maximize

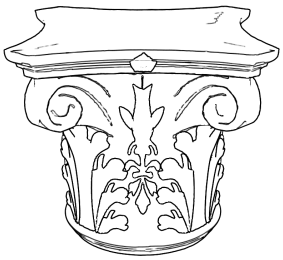
view dependent curvature as illustrated in Figure 2.3.9b. Demarcating curves [51] are points where the surface presents the "strongest" inflection, i.e. the surface changes its convexity most markedly at these points. Intuitively, they are present between valleys and ridges since they correspond to concave and convex regions on the surface, respectively. Mathematically, they are defined as the set of points where the second fundamental form reaches zero as exemplified in Figure 2.3.9c. Laplacian curves [86] are a set of points on a surface where the Laplacian of illumination reaches zero and the magnitude of the gradient of illumination is above a small threshold. Laplacian curves, therefore, capture surface features with respect to an external light source as isophotes do. Figure 2.3.9d gives an example.



(a) Creases (Saito and Takahashi [74])

(b) Apparent Ridges (Judd *et al.* [42])

(c) Demarcating Curves (Kolomenkin *et al.* [51])



(d) Laplacian Lines (Zhang *et al.* [86])

Figure 2.3.9: Various Types of Lines

The lines discussed so far are usually used in visualization applications where there is only a single layer of surfaces, which loosely refers to applications that involve only opaque geometries. These rendering tasks also heavily utilize regular geometric shading and lighting, both of which reveal the shape of the geometry significantly. As a result, the lines in these applications are

used primarily for enhancing shape visualization, emphasizing certain lines for illustrative purposes, simulating human hand drawing or doing stylistic rendering. Such applications constitute a large portion of geometrical visualization applications.

There are, however, other visualization tasks that require display of multiple (usually two) layers of surfaces or shapes that overlap spatially or become overlapped when they are projected onto the virtual view plane. For example, in the study of strokes, researchers have to understand the spatial relationships between the damaged portion and the rest of the brain as the place and shape of the damage is of crucial importance to function loss. In planning cancer operations, radiologists must determine the spatial structure and location of the target tumor relative to surrounding organs. All of these examples can be abstracted into the same visualization problem, i.e. how to display multiple overlapped surfaces in a way that allows humans to understand each of them as well as their spatial relationships. To make them both visible in a 2D display, it is necessary to display the outer surfaces with some degree of transparency, at least in the overlapped region. The effectiveness of this visualization, however, is significantly impaired due to loss of contrast and visual confusion, making comprehending these shapes a challenging task.

As a result, visualization experts have focused on how to utilize line generation techniques to facilitate perception tasks in these applications, the first one being Interrante *et al.* [37], who proposed applying sparsely distributed shape-telling lines on transparent surfaces to help reveal the shapes, testing this technique in an operation planning experiment. In her setup, there were usually two geometries, one completely contained by the other. Thus, a natural display configuration was to render the containing surface translucently and the contained one opaquely. She then added such auxiliary markers as dots, strokes, lines following principal curvature directions, and ridge and valley lines to cover the containing surface in rendering [34, 35, 36, 37]. She conducted several controlled experiments, verifying that these lines significantly improved the shape perception. She also concluded that lines following principal curvature directions are among the most effective in improving shape perception.

Although there are good reasons to believe that curvature following lines are effective in revealing shape, this does not mean that non-curvature related lines are not functional. In fact, Interrante *et al.* [34] showed that a simple grid projected onto the geometry seems to be as effective as principal curvature direction lines, although no firm conclusion was drawn due to insufficient samples. House's group [6, 31], on the other hand, was particularly interested in evaluating the effectiveness of

non-curvature based lines in conveying shape of terrain-like geometries. Bair *et al.* [6] designed and evaluated several types of such lines in an overlapped visualization application. In their experiments, they displayed two terrain surfaces simultaneously, with one on top of the other as exemplified in Figure 2.3.10. The top terrain surface was assigned some transparency to allow visibility of the

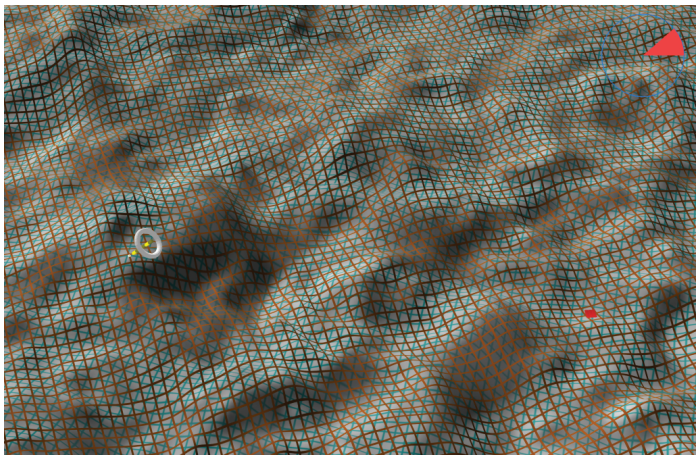


Figure 2.3.10: Line Texture Enhanced Overlapped Surfaces Rendering (Used by Permission)

bottom one. The major difference separating their work from others was their application of grid lines of varying parameters on both surfaces. They also conducted a thorough search of the parameter space using human controlled data mining techniques, a process which identified important factors affecting the ability of lines to reveal shape [31]. Their results demonstrated that color is of least importance and that opacity of the top terrain surface was most effective at approximately 0.4. The most striking conclusion, however, was that a pair of grid lines covering two overlapped terrain surfaces was the optimal line combination in enhancing human perception of the covered terrain surfaces. This effect is especially strong in a stereoscopic display environment.

This chapter covered several aspects of descriptive line elements, including the underlying theory, their mathematical properties, their generation techniques, and their implications and applications in visualization tasks. It also examined the literature of enhanced perception visualization techniques, especially those involving overlapping surfaces. The major conclusion is that lines are effective in conveying shape information, including curvature based and grid lines. Finally, it laid down the necessary foundation of volume rendering and descriptive line elements. The main goal of this research, however, is to combine these two techniques to allow enhanced perception in a volume rendering process. The necessary implementation and user study to validate the approach reported

here are discussed in the next chapter.

Chapter 3

Projected Grid Textures

The focus of the work reported here was to combine volume rendering and descriptive line elements and extend them to allow enhanced human perception of complex shapes involved in a volume rendering process. Such a rendering technique will benefit volume rendering applications involving overlapping objects as it is important to understand the spatial relationship among them, especially critical in medical volume visualization. One such example is in the study of strokes, where clinicians must be able to understand the relationship between the damaged portion of the brain and the surrounding anatomy in one rendering. This understanding is difficult to achieve in a volume rendering that relies only on transfer functions because object overlap in the projection impacts the perception of shapes as they become visually confounded. Opacity of the objects can be adjusted to allow clearer visualization of any one of them, but this is at the expense of making other objects even harder to visualize. For example, making a foreground object more opaque occludes background objects, while making it more transparent reveals the background but at the expense of foreground contrast.

Therefore, this approach is not a practical solution to the problem. What is needed is to keep the opacities at appropriate levels such that all objects are reasonably visible in the rendering but to provide auxiliary visual cues to enhance the clarity of surface shape without adding too much visual disturbance. Various desirable features of these added visual cues are listed below:

- They must be visible enough to help.
- They must not be so visible as to cause occlusion of or confusion with features in the data.

- They need to convey object shape information.
- They should be integrated into the rendering process.

These requirements and features lead to the consideration of descriptive line elements. As introduced in Chapter 2.3, existing research has suggested that these lines convey the shape information of the covered surfaces. Moreover, their density and location on the surface are adjustable, as are their visibility and occlusion of the underlying surfaces. There is, of course, a tradeoff between their coverage and occlusion of the surface. Nonetheless, they can satisfy the first three features, leaving incorporation of these lines into the volume rendering process as a key goal of this research.

One more problem, however, still remains; i.e. there is no clear definition of surface in a volume rendering process. As explained in Chapter 2.2, volume rendering concerns light medium interactions over a continuous domain of the signal. It relies on transfer functions to map the volume data to optical properties for rendering. Thus, it does not require the definition or existence of surfaces, making it suitable for describing many optical phenomena in nature that surface-based graphics cannot capture. Descriptive line elements, on the other hand, are based on the differential properties of surfaces. Even projected textures, like the various grid patterns investigated by House's group in their studies, still require a parameterized surface for which these lines are generated and onto which they are applied. To bring these two techniques together, the research here relies on the heuristics that the objects which interest people in a volume rendering process ultimately have shapes, like the brain in the medical visualization example, and that an isovalue defines an isosurface in a volume dataset. Thus, the boundary of shapes of interest can be considered as a surface defined with respect to an isovalue, and, therefore, descriptive line elements can be designed, generated and applied onto the boundary. This understanding makes sense even for irregular objects, such as vasculature or tumors, because the real objects have shapes and the boundary of these shapes can be defined with respect to an isovalue. Based on these concepts, this research extends volume rendering with descriptive line elements and conducts necessary experiments to examine if this approach enhances human perception. To achieve this goal requires a fast volume renderer that is fully controlled and to which non-trivial modifications can be made to implement the required extensions. In addition, experiments must be designed and conducted to yield quantitative measures that can be collected and analyzed statistically so that the results can be verified and studied.

3.1 Integrating Texture into a Volume Renderer

This section describes in detail the volume renderer implementation that forms the basis for this research. Chapter 2.2 gave theoretical explanations about volume rendering as well as an overview of several algorithms. There is, however, a difference between theory and implementation. As a result, several important factors need to be considered and corresponding decisions made based on issues not covered by theory. One major factor is that volume rendering is a computationally intensive process. Therefore, it relies heavily on high performance computing techniques and hardware.

As the experiments for this research were conducted in a stereoscopic display environment, it is easier to introduce the hardware configuration first. Figure 3.1.1 displays the machine used for these experiments. As this figure shows, Two IBM T221 "Big Bertha" LCD monitors are mounted

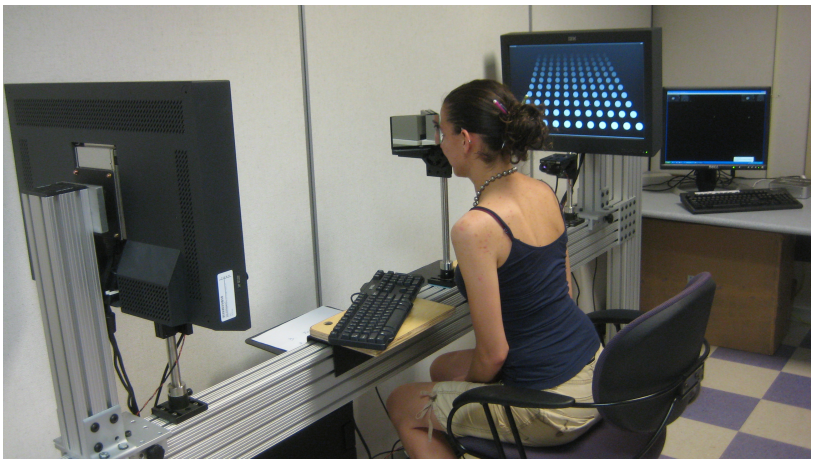


Figure 3.1.1: Stereoscopic Display Setup

on two sides of a supporting track. In the middle of the track, a pair of front surface mirrors are placed in such a way that the left mirror reflects the left screen into the left eye and the right mirror reflects the right screen into the right eye in a Wheatstone stereoscope arrangement [83]. The screens are 48 cm in width and 30 cm in height, giving an aspect ratio of 16:10. Each screen has a native resolution of 3840×2400 pixels, or 9.2 million pixels and the physical size of a pixel is 0.01245 cm^2. For the experiment conducted here, the screens were set at a viewing distance of 86.36 cm away from each corresponding mirror. At this distance, one screen pixel corresponds to 0.5' of the visual angle. This is at the resolution of the human fovea, which is approximately 60 cycles per degree, corresponding to 1 cycle per minute of visual angle [9]. The monitors, therefore, introduce virtually

no visible pixel level artifacts. The screen contrast ratio is 400:1 with a maximum brightness of $235\,\mathrm{cd/m^2}$. The two monitors are driven by two NVIDIA Quadro FX 5800 graphics cards, each equipped with 4G memory and 240 computing cores. One card supports the left half of both screens and the other the right half as illustrated in Figure 3.1.2. This design allows for automatic support

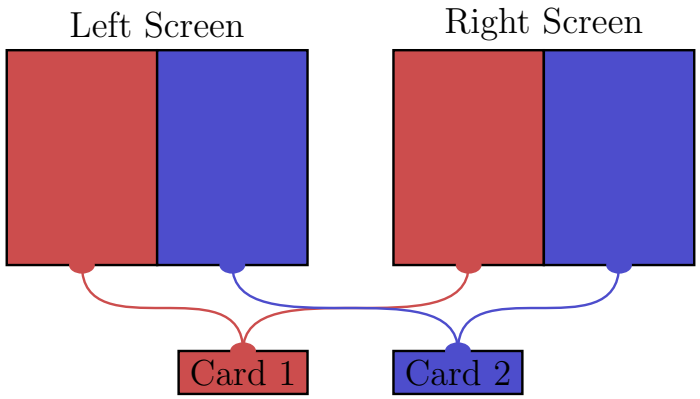
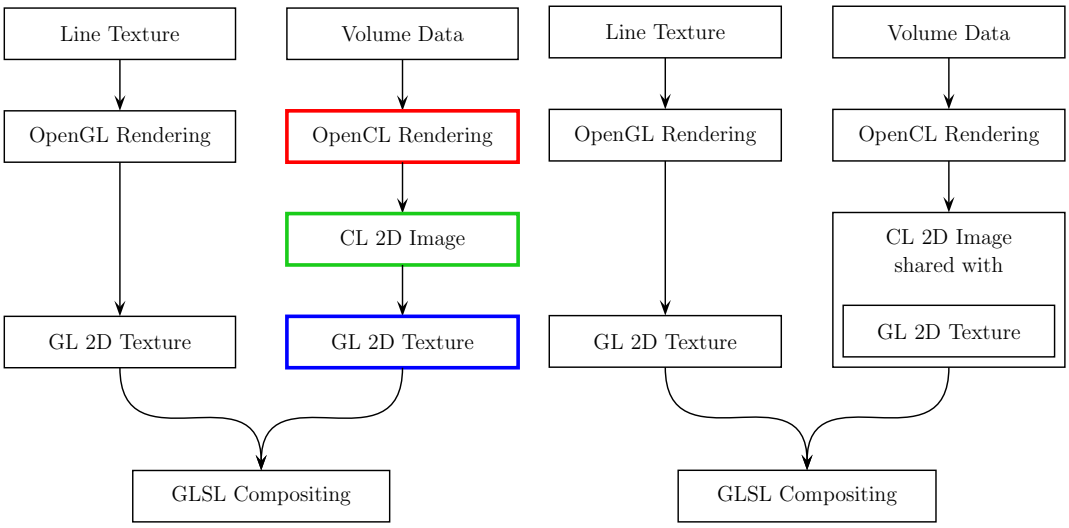


Figure 3.1.2: Stereoscopic Screen Setup

of the stereo effects of most OpenGL applications. The video cards are programmable, with their driver also providing native support for GPGPU programming interfaces like CUDA and OpenCL as well as for GLSL programming and fast interpolation for 3D textures implemented at the hardware level. These cards and their supported features formed the hardware foundation for the volume renderer implementation done here.

To utilize the computing power of the graphics cards, it is necessary to choose an appropriate programming interface, one through which a host program can obtain access and control of the resources on the cards. Utilizing such graphics hardware to perform accelerated rendering is generally known as GPU programming. GLSL [72], for example, is an extension of OpenGL [84] designed to allow an OpenGL program to complement the fixed rendering pipeline as well as to more flexibly harness the computing power provided by the graphics hardware. A disadvantage of GLSL is that, since it is designed to complement the OpenGL rendering pipeline, it has several limitations affecting general computational tasks. Other interfaces are designed for more general purpose computation. These alternatives and the computational models that they nurture are referred to as General Purpose GPU Programming (GPGPU). Since they are usually designed with a broader focus than graphics computation, they have no inherent concept of rendering or graphics; instead they focus on native support for the mathematical operations essential for all computations. As

they emerged later in the history of GPU development, they also enjoy a clean design that maps almost directly to current GPU hardware architecture. These features allow for easy manipulation of the underlying hardware and intuitive reasoning and implementation of most algorithms. Two of the most widely used GPGPU programming interfaces supported by graphics hardware are NVIDIA CUDA [67] and OpenCL [45]. CUDA is a proprietary parallel computing architecture developed and maintained by NVIDIA with active development and support. As NVIDIA is still playing a leading role in GPGPU computing hardware design and manufacture, CUDA enjoys a wide range of applications. Its disadvantage, however, is that it is proprietary and as such is only supported on NVIDIA hardware. In contrast, OpenCL is designed as an open specification and framework supporting GPGPU programming and computing across heterogeneous platforms consisting of CPU, GPU and other processing units. It was developed and is maintained by the non-profit technology consortium Khronos Group [29] with active participation and support from major hardware manufacturers. For the research conducted here, an open standard, which allows for the future opportunity of moving to non-NVIDIA hardware to provide maximum maintainability, was chosen. Therefore, it used OpenCL as the GPGPU computing framework in its volume renderer implementation.



(a) Two-Card Configuration (b) Improvement for One-Card Case

Figure 3.1.3: Overview of Rendering Pipeline

As a volume renderer is a graphics application, OpenGL was also used here to facilitate implementation. Figure 3.1.3a shows the flow of the rendering pipeline. The actual volume rendering algorithm is implemented in OpenCL (Figure 3.1.3a red box), and the output is directed

into a buffer (Figure 3.1.3a green box) that is copied into OpenGL (Figure 3.1.3a blue box) for further manipulation and display. Since GPGPU computing emerged from and was inspired by GPU computing, the design of OpenCL facilitates cooperation with existing graphics interfaces like OpenGL. In particular, the OpenCL specification allows an OpenCL context to be constructed from an existing OpenGL one, given additional runtime arguments to identify a display environment to which the latter is attached. In such an inter-operation setup, it is possible to share memory objects between OpenCL and OpenGL. The performance implication is that the per frame copying of the rendering buffer from OpenCL to OpenGL can be completely eliminated by sharing the same memory object between the two as illustrated in Figure 3.1.3b, thus allowing faster rendering and higher frame rates. However, our implementation had to deal with additional limitations due to the two-GPU-two-display architecture used. At this time, OpenGL-OpenCL interoperation seems to work only for a single physical card configuration. Since the stereoscopic display environment here was driven by two physical cards, both of which will be utilized in rendering, a less efficient solution of copying the render buffer every frame was chosen.

For the reasons indicated in Section 2.2.3, a ray marching algorithm was implemented as the volume renderer in this research, a choice based on the target data structure as well as the capability of the hardware used here. The ray marching algorithm is embarrassingly parallel because it requires virtually no synchronization between different rays. Therefore, each ray can be processed separately and simultaneously by a single thread. In this sense, ray marching maps almost directly to the graphics hardware threading model. In the implementation used here, a thread was launched for each ray, allowing for easy management. One potential variation uses persistent threads [3], where the number of threads launched is the same as the number of single processors of the graphics hardware. In this model, each thread is kept alive for as long as possible and recycled to process the next ray in the pool when it is finished with the current one. Thus, the overhead of launching and terminating threads is kept to a minimum. The benefit of the persistent thread technique, however, is subject to the hardware thread management expense. On the graphics hardware used here, no significant performance enhancement was observed using this technique.

As a result, this research used an intuitive and simple thread management model. In it, each ray is intersected with the bounding box of the domain of the volume data. Further ray marching is performed only if the ray hits the bounding box. When a ray hits, it uses a fixed sample distance to step through the volume. At each sample position, trilinear interpolation is used

to query the volume data for a value representing the signal at the sample position. This value is used to index pre-computed transfer functions to provide an opacity value and an RGB color value for compositing. The opacity and color are subject to necessary adjustment if the actual sample distance is different from the sample distance used to compute the transfer functions as was explained in Chapter 2.2. When lighting is enabled, the gradient vector at the sample position is computed using central difference based on Equation 2.2.8. For boundary grid points where central difference is not possible, forward difference or backward difference is employed. It is very important to produce artifact-free rendering as our experiment was a perception study and any artifacts from rendering might interfere with human perception with unpredictable results. This situation is especially true with the stereoscopic displays used here because a very high resolution is more likely to expose rendering artifacts that are usually invisible at a coarser one. Therefore, virtual sampling was used to obtain a rendering quality produced at one-fourth of the current step size, resulting in approximately a 2 to 3 fold increase in speed with enhanced rendering quality.

Our technique focuses on the incorporation of a line texture in volume rendering, currently achieved through multiple separate renderings and additional composting. The line texture was stored as separate geometrical objects and rendered by OpenGL without lighting. The result was directed into a GL 2D texture. The result of the volume rendering was also copied into a GL 2D texture. In the end, a GLSL shader was used to composite the line texture with the volume rendering into a final frame for display. Figure 3.1.3a shows this pipeline, and Figure 1.2.1 shows the decomposed renderings at different stages. This pipeline works well for a one volume object and its accompanying line texture.

However, the work reported here also requires the study of the effect of line textures in an overlapped visualization situation, thus requiring using complex transfer functions to select two objects in volume rendering. This experiment used a more translucent transfer function for an external object in the volume data and a more opaque transfer function for an internal object, each one having its own accompanying line texture. The line texture was rendered by OpenGL, with textures from different objects being rendered separately. This process required a complex rendering pipeline because of the need to separate different layers properly to ensure correct compositing. This implementation rendered each line texture into a separate GL 2D texture. The volume renderer was also modified to produce two outputs, with one recording the color produced by the external layer of the overlapping volume objects and the other recording the internal layer. In the end, a GLSL shader

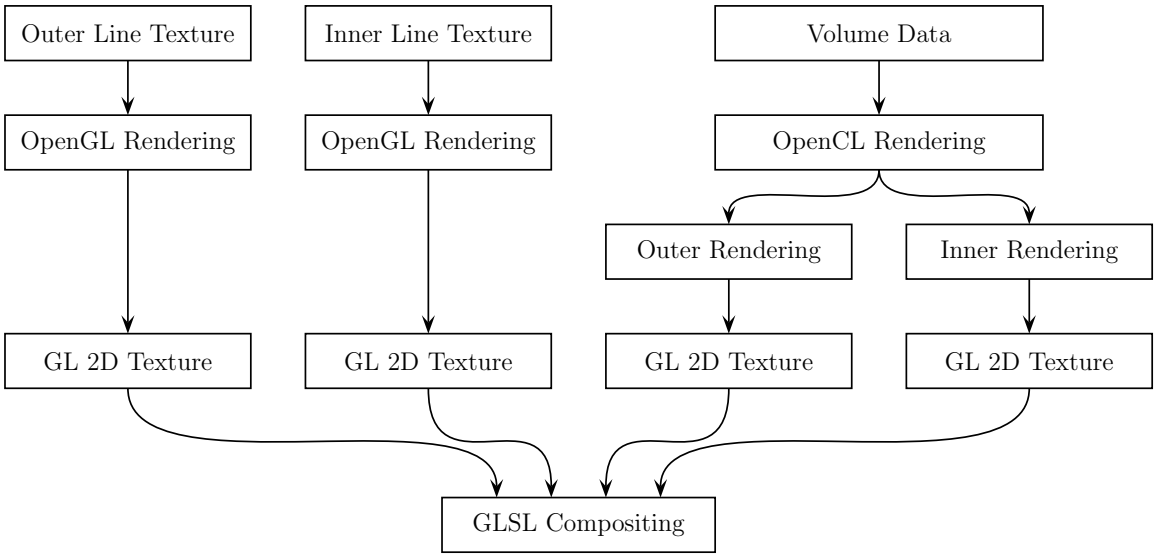
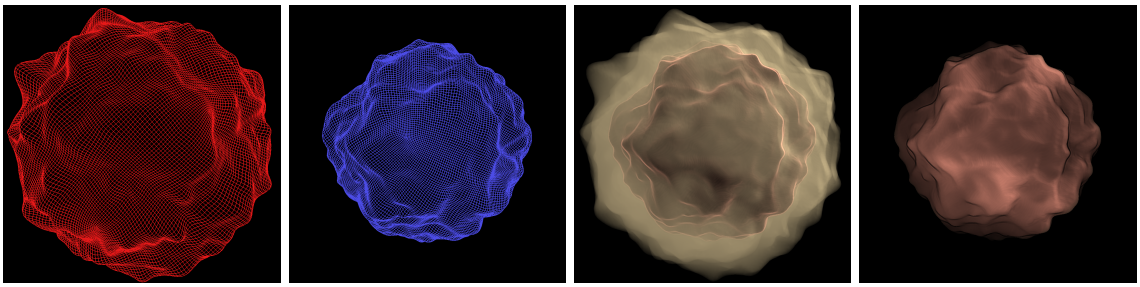
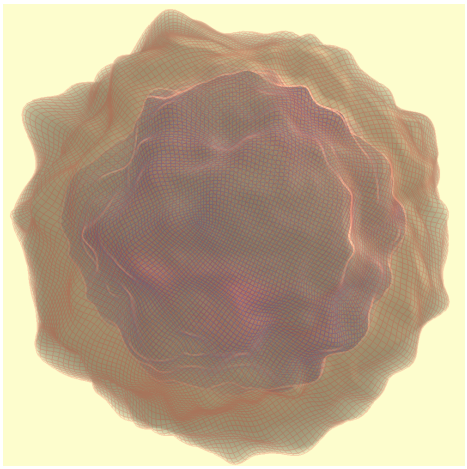


Figure 3.1.4: Layered Volume Rendering Pipeline

was used to composite the four layers along the view direction to produce the correct rendering for display. Figure 3.1.4 shows the flow of the pipeline. Figure 3.1.5a-d further shows the decomposed renderings at different stages of the pipeline, and Figure 3.1.5e shows the final rendering of a layered volume dataset.



(a) Outer Line Texture (b) Inner Line Texture (c) Outer Volume Only (d) Inner Volume Only



(e) Final Rendering of Layered Volume

Figure 3.1.5: Decomposition of Rendering Pipeline

3.2 Generating Projective Grid Textures

Having a renderer that incorporates line textures with volume rendering allowed us to design a user study to evaluate whether line textures are able to enhance perception in a volume rendering process as it is known that they do in a surface-based rendering model.

3.2.1 Study Goals

To evaluate this effectiveness, we designed user experiments to give quantitative results about perception related performance. Our studies analyzed two cases:

- The first case investigated if line texture added onto a single opaque object was able to enhance human perception in volume rendering. Although this setup is basic, it appears not to have been studied before.
- The second case used two objects with one nested inside the other. A transfer function assigned translucency to the containing object, thus allowing visibility of the contained object which was given full opacity. Line textures were generated for them, with a set of predefined opacity combinations being studied to determine which works best.

Shape perception was tested by asking subjects to orient a probe attached to the surface so that it matched the local surface normal. Each subject encountered a series of probe positions randomized across the two surfaces. Figure 3.2.1 illustrates this configuration, with the white line marking the normal direction and the bottom sphere the probe attachment position. This approach for estimating normal direction has been used in a number of perception studies in surface-based graphics [4, 6, 35].

3.2.2 Design of Irregular Shapes

The ultimate goal of this research is to enhance perception in practical visualization applications. However, as the first step in designing experiments to evaluate if the line texture approach is effective in volume rendering, it is advisable to have full control of the objects displayed and to ensure they are understood, both visually and mathematically. Therefore, we used carefully designed shapes for our perceptual experiments. The following guidelines were used to design the shapes:

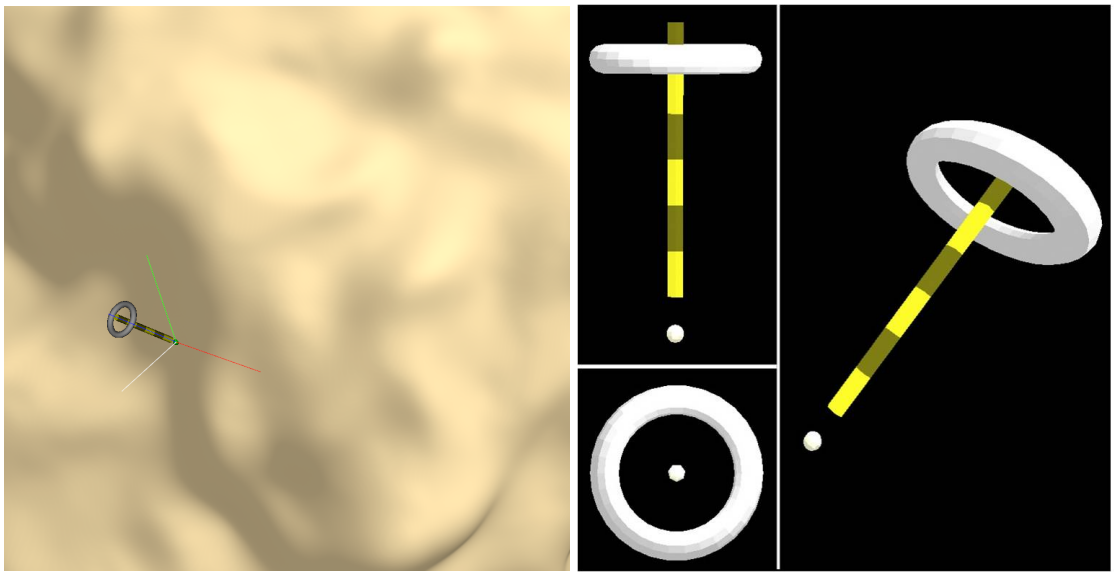


Figure 3.2.1: Probe

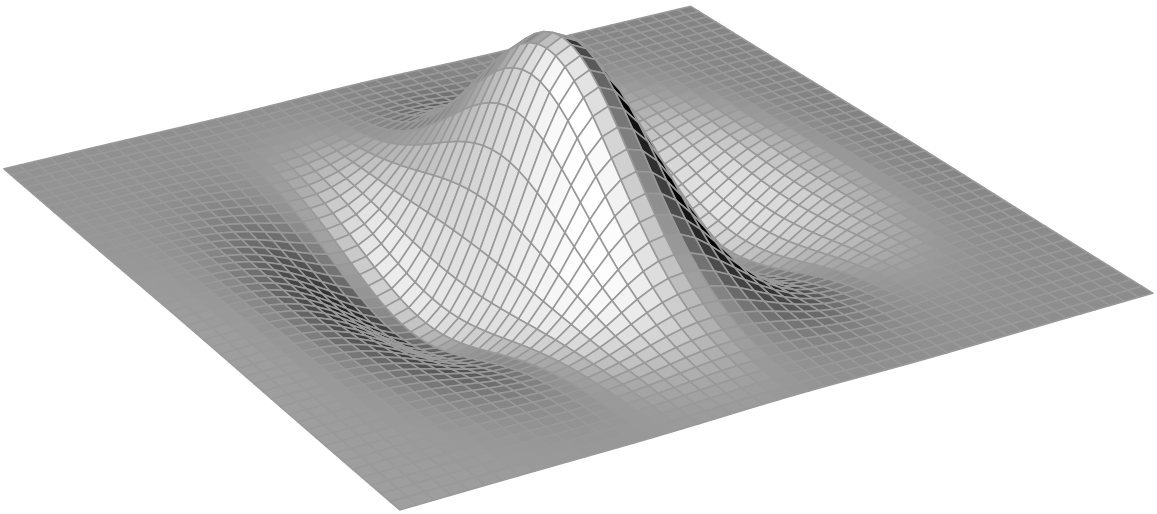
- The shapes need to be irregular. Otherwise lighting and common intuition can guide the participants to provide a good estimate of normal directions.
- The shapes must be somewhat smooth because otherwise it is too difficult to determine the surface normal direction.
- The shapes need to include random features so that different shapes for different users can be generated to eliminate the system bias that can lead to a predetermined yet unknown result.
- The randomness of the shapes must be statistically uniform so that data from different users can be compared fairly.

The goal of the experimental study was to examine the effectiveness of this technique in a volume rendering process. To make the irregular shapes closer to those usually found in volume rendering, organic-looking shapes that have self-similarity at different scales [61] were generated by adding perturbation kernels of different magnitudes and random parameters to a regular background shape. Bair and House [4], using this approach, added random Gabor bumps onto two planes to produce irregular terrain surfaces, subsequently using them to investigate the effectiveness of line textures in revealing their shape in an overlapping visualization situation. Our study employed a similar approach, using the sphere as the basic underlying shape. Gabor bumps of different magnitudes were added onto it to form the irregular shapes, the Gabor surfaces providing a different look to the

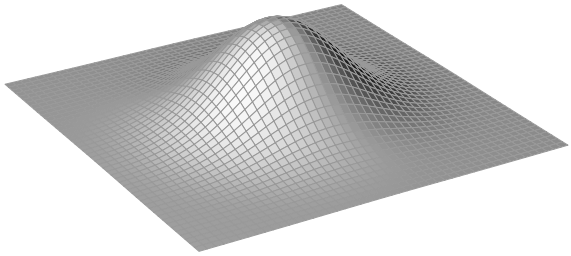
sphere, while mathematically providing an explicit and analytical equation that described the final surface. The Gabor function

$$g(x, y) = A \cos\left(\frac{2\pi}{T}x\right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2.1}$$

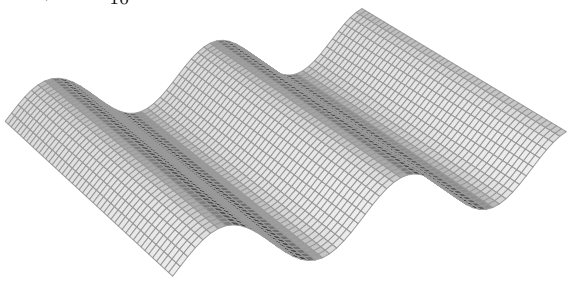
is a 2D Gaussian function modulated by a 1D cosine function, with the amplitude parameter A



(a) $A = 3$, $T = 2\pi$, $\sigma = \frac{3}{10}T$



(b) Gaussian Component

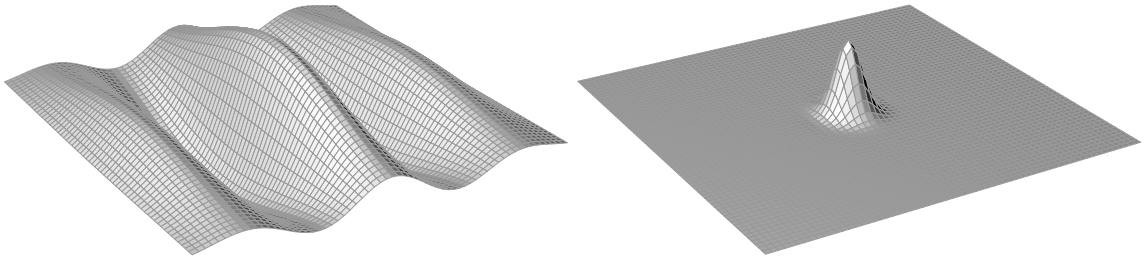


(c) Cosine Component

Figure 3.2.2: Gabor Function and Its Components

controlling its overall magnitude. The Gabor function defines a surface in three dimensional space that has a look similar to that illustrated in Figure 3.2.2. The cosine function and the Gaussian function together control the shape. Given different parameter sets, the Gabor surface can have quite different looks. For example, Figure 3.2.3 shows Gabor surfaces produced by two sets of parameters.

For our work, we tuned the Gabor functions to produce only one valley on each of its tails so that the surface generated has ridges as well as valleys. This requirement implies that the cosine function and the Gaussian function must meet the condition shown in Figure 3.2.4, meaning the



(a) $A = 3$, $T = 2\pi$, $\sigma = \frac{3}{4}T$ (b) $A = 3$, $T = 2\pi$, $\sigma = \frac{1}{10}T$

Figure 3.2.3: Gabor Function with Different Parameters

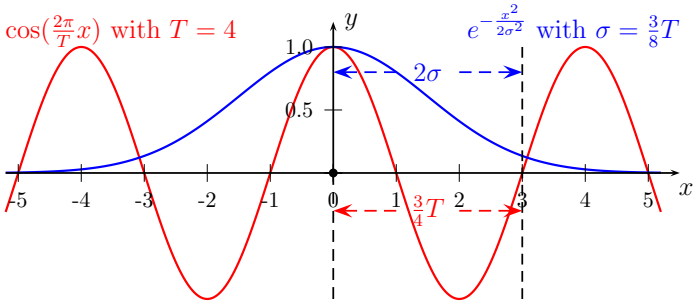


Figure 3.2.4: Cosine and Gaussian Match

period T of the cosine function and the σ parameter of the Gaussian function need to satisfy

$$T = \frac{8\sigma}{3}$$

making the Gaussian function fade at approximately the first complete valley of the cosine function. In practice, however, it was found that

$$T = \frac{10\sigma}{3} \tag{3.2.2}$$

produced Gabor bumps that provided a smoother look to the irregular shapes. Since the Gaussian function's amplitude A is usually greater than $\frac{1}{2\sigma^2}$, there is still some visible amplitude at 2σ position from the Gaussian component; stretching the cosine function slightly allows the Gaussian to fade more, thus producing smoother tails.

The sphere function is defined on a 3D domain while the Gabor function is defined on a 2D domain, resulting in an inconsistency in dimensionality, which can be easily addressed by ignoring one dimension of the sphere for the Gabor functions, for example the z dimension. Doing so causes this ignored dimension to become an extra degree of freedom for the Gabor function with respect to the sphere function. This situation can be utilized to allow for increased random deformation on

the sphere surface caused by the Gabor surface, accomplished by rotating the axis of symmetry of the Gabor function away from the z axis of the sphere function by a random amount.

To perturb the sphere surface using the Gabor surfaces, the value of the Gabor function is used as a displacement for the sphere function in its radial direction. Figure 3.2.5 gives an example in 2D, in which a unit circle centered in the origin is perturbed by a Gabor function where $A = \frac{T}{10}$, $\sigma = \frac{3}{10}T$ and $T = 1$. Since the deformation is only applied along the normal direction, the topology

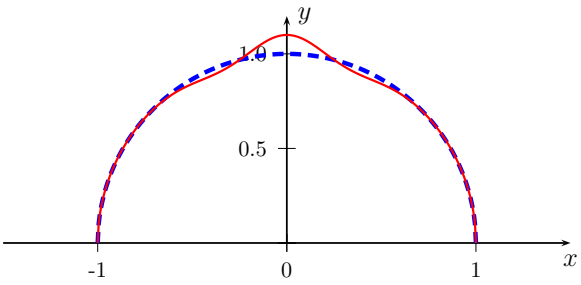


Figure 3.2.5: Gabor Added Onto a Circle

of the tessellation of the sphere surface is not changed, a desirable outcome. The construction of the displaced surface is represented by

$$\begin{aligned} f\left(x,y,z\right) &= \left(x^2+y^2+z^2\right)^{1/2} - r - \textstyle\sum_{i=1}^{n} g_i\left(x,y,z\right) \\ g_i(x,y,z) &= \tilde{g}_i\left[R_i(x,y,z)^T\right] \\ \tilde{g}_i(\tilde{x},\tilde{y},\tilde{z}) &= \begin{cases} A_i \cos\left(\frac{2\pi}{T_i}\tilde{x}\right) e^{-\frac{\tilde{x}^2+\tilde{y}^2}{2\sigma_i^2}} & \tilde{z} \geq 0 \\ 0 & \tilde{z} < 0 \end{cases} \end{aligned} \tag{3.2.3}$$

where r is the radius of the sphere, g_i the i^{th} Gabor function and A_i, T_i and σ_i its parameters that are randomly generated. The R_i is a random rotation applied onto the Gabor function to give it an orientation with respect to the z axis of the sphere. The irregular surface is determined by Equation 3.2.3, making it an implicit surface with properties that can be determined analytically. One such property, for example, is that the gradient of f is given by its first order partial derivative with respect to x, y and z in the analytical form as

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{x}{(x^2+y^2+z^2)^{1/2}} - \sum_{i=1}^{n} \frac{\partial g_i}{\partial x} \\ \frac{y}{(x^2+y^2+z^2)^{1/2}} - \sum_{i=1}^{n} \frac{\partial g_i}{\partial y} \\ \frac{z}{(x^2+y^2+z^2)^{1/2}} - \sum_{i=1}^{n} \frac{\partial g_i}{\partial z} \end{bmatrix} \tag{3.2.4}$$

where

$$\begin{bmatrix} \frac{\partial g_i}{\partial x} \\ \frac{\partial g_i}{\partial y} \\ \frac{\partial g_i}{\partial z} \end{bmatrix} = R_i^{-1} \left(-A_i e^{-\frac{\tilde{x}^2+\tilde{y}^2}{2\sigma_i^2}} \right) \begin{bmatrix} \frac{\tilde{x}}{\sigma_i^2} \cos\left(\frac{2\pi}{T_i}\tilde{x}\right) + \frac{2\pi}{T_i} \sin\left(\frac{2\pi}{T_i}\tilde{x}\right) \\ \frac{\tilde{y}}{\sigma_i^2} \cos\left(\frac{2\pi}{T_i}\tilde{x}\right) \\ 0 \end{bmatrix}$$

and $R_i^{-1} = R_i^T$ is the inverse rotation represented by R_i, and $(\tilde{x}, \tilde{y}, \tilde{z})^T = (x, y, z)^T R_i^T$. Moreover, based on Equation 3.2.3, it can be concluded that $\nabla g \equiv 0$, $\forall \tilde{z} < 0$.

Since we required random surfaces sharing statistical properties, guidelines are needed for choosing parameters that give the desired statistical properties. As Equation 3.2.2 gives the relationship between the cosine function and the Gaussian function, the σ parameter effectively controls the span of the Gabor surface. Ideally, one Gabor surface should cover an amount of the sphere surface area so that it is easily visible but not too large. Through experimentation, it was found that the largest Gabor bump should span approximately 60° on the sphere surface, requiring

$$\hat{\sigma} = \frac{r \tan(30°)}{2}, \tag{3.2.5}$$

where r is the radius of the sphere. Also through experimentation, the amplitude of the Gabor was found to be best at 1/10 of the period of the cosine function, i.e., $A = T/10$.

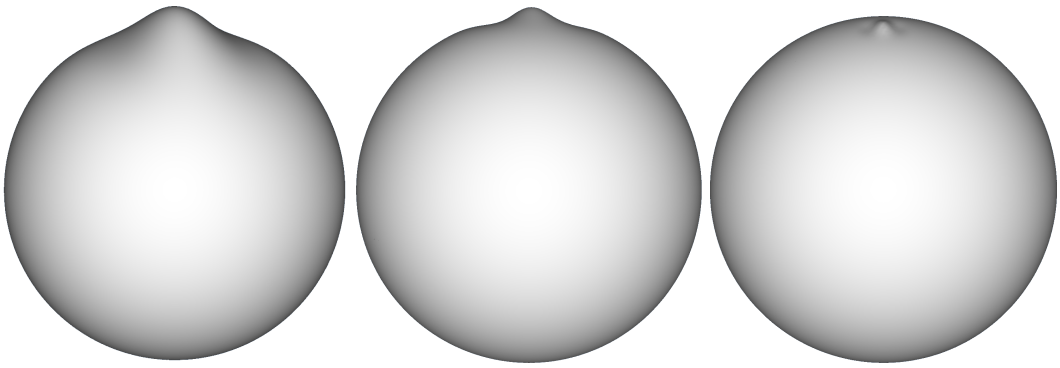
To add the randomness, the σ parameter of the Gabor function was drawn from a Gaussian distribution with a mean of $\hat{\sigma}$, as defined in Equation 3.2.5, and a standard deviation of $0.08\hat{\sigma}$. In addition to these parameters, the Gabor function was given a random orientation with respect to the z axis of the sphere function, realized by randomly picking a direction in the 3D space to which the Gabor function was oriented. This direction was composed of three values drawn from a $[0, 1]$ uniform distribution. The disadvantage of this method is that the unit box corner region has a higher chance of being selected. To address this problem, a Poisson disk sampling [62] inspired method was used to control the placement of a randomly generated Gabor surface. Specifically, any candidate Gabor function was discarded if its center fell within 30° of any of the Gabor functions already generated. This constraint still allows two Gabor surfaces to overlap since the center is being controlled.

To add more randomness to the generated surfaces, three magnitude levels were used. The parameters for the first level are given above. For every succeeding level, the number of Gabor functions was three times its preceding level, the repulsive degree between them one half and the σ

value one third. Their relationships are represented by the following equations:

$$\begin{aligned} \sigma_{n+1} &= \tfrac{1}{3}\sigma_n \\ D_{n+1} &= \tfrac{1}{2}D_n \\ N_{n+1} &= 3N_n \end{aligned} \tag{3.2.6}$$

In Equation 3.2.6, D_n and D_{n+1} are the repulsive degrees between Gabor functions from succeeding levels, and N_n and N_{n+1} the number of Gabor functions placed. While mutual exclusion was enforced for all Gabor functions of the same level, these from different levels, however, were free to overlap on the sphere surface. There were 30 Gabor functions in the first level. The constant coefficients in Equation 3.2.6 mimic the fractal geometry pattern found in nature [61]. Figure 3.2.6 shows the relative magnitude of the Gabor functions with respect to the underlying sphere function.



(a) First Level Gabor (b) Second Level Gabor (c) Third Level Gabor

Figure 3.2.6: Gabor Functions of Different Amplitudes (Visualization by ParaView [33])

For the overlapped visualization experiments, two irregular shapes were required with one containing the other, meaning two sphere surfaces of different radii were used as the starting geometry for shape construction. However, there is the possibility that the surfaces generated may intersect in some region due to over perturbation. Figure 3.2.7 gives an example of this problem in 2D where the dashed lines are the base functions perturbed by the Gabor functions. Geometrically, this situation is fine because each surface still maintains the correct topology. However, when these two shapes intersect, their visualization becomes unclear because not only the translucency but also the unexpected intersection contributes to the confusion. Since this study primarily focuses on evaluating the effectiveness of the line textures in translucent visualization applications, other interfering factors need to be eliminated. Therefore, the generated surfaces should not intersect,

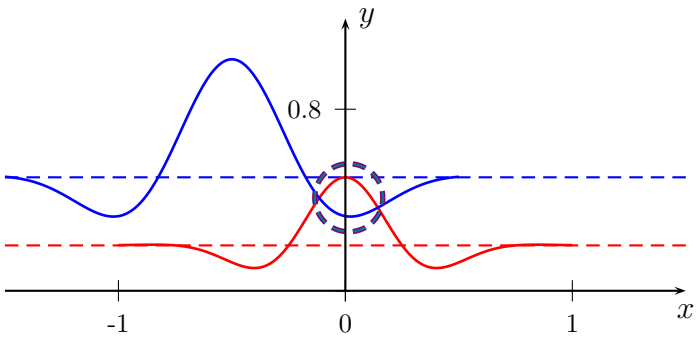


Figure 3.2.7: Gabor Perturbation can Cause Intersection between Two Surfaces

a situation that can be easily achieved by having a large ratio between the radii of the outer and inner spheres. However, this condition implies that in the visualization, the inner shape will be very small, a situation made even more so by the perspective projection. Having a small projection is not ideal for perception performance verification for two reasons. First, humans are not good at identifying details of small objects. Second, a small projection also means less pixel coverage, making it impossible to display details. The problem is further complicated by this confusing visualization, again resulting in uncontrollable factors that can interfere with the experiments. Therefore, what is needed is a pair of radii for the spheres such that they are separated enough so that the shapes generated do not intersect but close enough so that the projection of the inner shape is reasonably large to allow sufficient visibility. These conditions were achieved by choosing 10 cm as the radius of the inner sphere and 13 cm for the outer one with respect to the size of the virtual screen which was modeled after the physical configuration of the screen. As introduced in Chapter 3.1, each screen is 86.36 cm away from its corresponding mirror and the physical size of a pixel is 0.01245 cm^2. At a resolution of 2400^2 pixels, this gives a virtual screen of 29.88 cm^2 at 86.36 cm away from the eyes. Appendix A provides the reason for these choices.

3.2.3 Generation of Grid-like Line Texture

Equation 3.2.1 and 3.2.3 give a complete definition of the random surface generated. For the experiments, however, volume data were needed for rendering as are line textures. Both were achieved by using a constructive approach similar to the definition given in Equation 3.2.3 which includes the following steps:

1. Begin with a unit cube centered at the origin and apply Catmull-Clark subdivision [11] for

several iterations. Here six iterations were used as they give a reasonably fine quadrilateral tessellation of the subdivision surface, which almost assumes a spherical shape at this iteration. This process is illustrated in Figure 3.2.8.

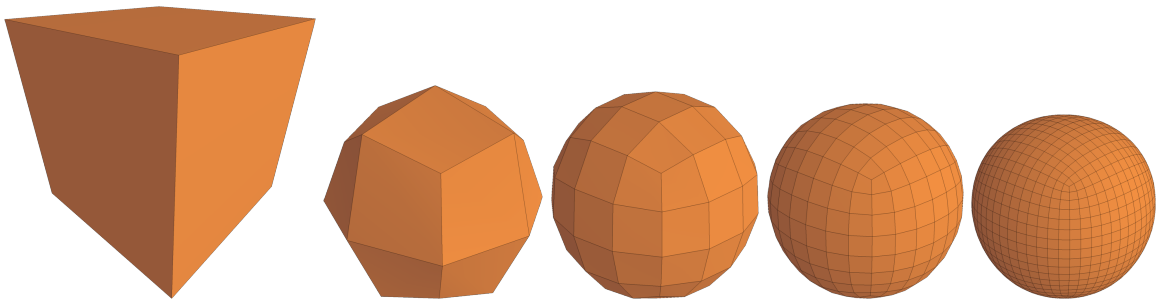


Figure 3.2.8: Catmull-Clark Subdivision of a Unit Cube (Generated by TopMod [25])

2. Find the vertex of the subdivision surface that is farthest from the origin. Record its distance to the origin and push all vertices of the subdivision surface onto a sphere of this radius. This process effectively constructs a grid-like quadrilateral tessellation of the sphere surface, with the exception of the vicinity of the eight vertices of valence 3, originating from the cube's corners.

3. For each vertex in the tessellated surface, compute the sum of displacement caused by all Gabor functions using Equation 3.2.3. This value is used to displace the vertex in the radial direction given by the normal direction at the vertex on the sphere.

The result of these steps is a quadrilateral tessellation of the surface defined by the mathematical equation for a particular set of randomly generated Gabor functions. The edges of the quadrilaterals formed the grid-like line texture needed as seen in Figure 3.2.9. Only the edges were kept as geometrical primitives for rendering. In the overlapping visualization experiment, these steps were completed for each random shape. As a result, there were two such edge-only geometries.

3.2.4 Generation of Volume Data

The next step was to produce the volume dataset. First, the underlying grid structure of the volume data was computed. The domain of the volume data needed to fully enclose the irregular shape. This was computed by finding the longest distance between the origin and the vertices of the tessellated irregular surface that is produced when computing the line texture for the same shape.

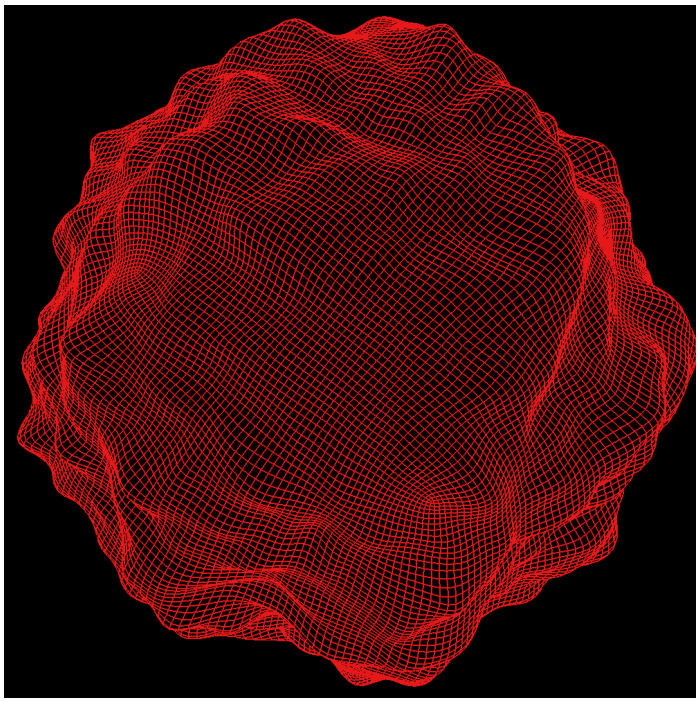


Figure 3.2.9: Quadrilateral Tessellation of Irregular Shape

Since the shape was built from a sphere function, this distance was then taken as the base size of the domain of the volume data because in all possible directions, the shape is bounded by this value. The domain was computed to be a cube with a side length that is twice the distance just computed.

Next, the voxel size was computed. As mentioned in Chapter 2.1, the value is governed by the Nyquist frequency for sampling the finest feature of the signal. For the application here, the finest feature was produced by the smallest Gabor function. Equation 3.2.1 gives the definition of the Gabor function used, the Fourier transform [17, 63] of which is

$$g_f(u,v) = 2\pi\sigma^2 A e^{-2\pi^2\sigma^2\left[\left(u-\frac{1}{T}\right)^2+v^2\right]} \tag{3.2.7}$$

indicating that g_f is non-zero everywhere on the 2D frequency domain. This result means that the function g in Equation 3.2.1 is not band-limited. Consequently, the overall shape is not band limited. Thus, it is impossible to sample the signal in a way to faithfully reconstruct it in a post processing step, meaning that whatever voxel size is used, tiny Gabor features will be lost. However, Equation 3.2.7 is also a Gaussian-shaped function in frequency with mean value $(\frac{1}{T}, 0)$ and standard deviation $\frac{1}{2\pi\sigma}$ for both dimensions. Therefore, when the frequency is cut at the 3σ position of this

function, then most of the information is expected to be preserved, indicating

$$\nu_s = 2 \times (\frac{1}{T} + 3 \times \frac{1}{2\pi\sigma}).$$

Based on Equation 3.2.2,

$$\nu_s = 3 \times (\frac{1}{5} + \frac{1}{2\pi})\frac{1}{\sigma}.$$

For the randomly generated smallest Gabor, this gives

$$\begin{aligned} \nu_s &\leq 3 \times (\tfrac{1}{5} + \tfrac{1}{2\pi})\tfrac{1}{(1-3\times 0.08)\times\hat{\sigma}\times\frac{1}{9}} \\ &< 3 \times (\tfrac{1}{5} + \tfrac{1}{2\pi})\tfrac{1}{0.75} \times 9 \times \tfrac{1}{\hat{\sigma}}. \end{aligned}$$

Equation 3.2.5 indicates

$$\nu_s \leq 72\sqrt{3} \times (\frac{1}{5} + \frac{1}{2\pi})\frac{1}{r}.$$

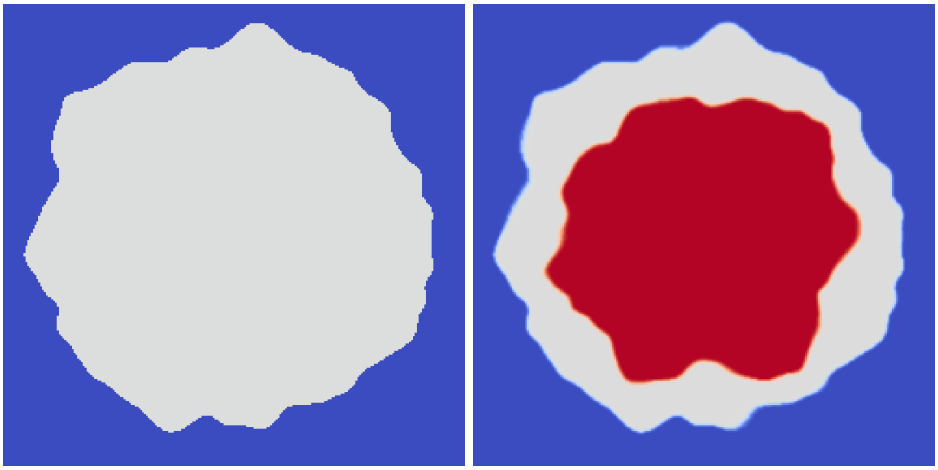
Therefore, the largest allowable sampling distance would be

$$s_M = \frac{r}{72\sqrt{3} \times (\frac{1}{5} + \frac{1}{2\pi})}.$$

This number was approximately 0.22 for the inner shape and 0.29 for the outer one. In practice, 0.1 was used because it allows the sampling process to remain within the upper bound established. The volume grid thus generated was approximately 250^3.

Knowing the domain size and the voxel size allowed for the computation of the underlying grid structure and the spatial positions of all grid points defined by the volume data. For each grid point, the function f in Equation 3.2.3 was evaluated, and the sign of the value was used to form a value for the current grid point with non-positive values mapped to 100 and positive values mapped to 0. This result produced a binary volume in which all grid points enclosed by the irregular surface have scalar values of 100 or 0, as illustrated in Figure 3.2.10a.

For the overlapping visualization study, two volume datasets were generated, one for the inner and one for the outer shapes. For the volume rendering process, these two datasets were merged into one. This merging operation was completed by iterating over all grid points of the outer volume and querying the inner volume for a scalar value at the same position. The queried value was then added to the value already stored at the grid point of the outer volume. This merging operation



(a) Volume Generated from Function (b) Merged Volume Data

Figure 3.2.10: Volume Data Generation (Visualization by ParaView [33])

thus produced a volume dataset that contained both irregular shapes. All grid points enclosed by both irregular shapes had a value of 200, while those enclosed only by the outer one had a value of 100, and the rest had a value of 0. Figure 3.2.10b shows one slice of the merged volume.

A further smoothing operation was applied to the volume data by using a Gaussian filter to smooth the sharp boundary, helping produce the high quality artifact-free renderings, essential for the experiments conducted here. While in a managed and constructive approach, it is possible to produce volume data that are smooth by nature, thus eliminating the need for a smoothing operation, in real applications, volume data represent objects in nature that do not necessarily have smooth boundaries. For example, in medical visualization, the data describe human body density which probably does not have smooth variations. Since this technique is intended to apply to those applications, it was beneficial to evaluate it on volume datasets, the properties of which are as close as possible to those in real volume datasets.

3.2.5 Generation of Experiment Parameters

Experiments were subsequently conducted to verify the effectiveness of this technique. In them, users were asked to estimate the normal direction at each of several positions on the boundary of the volume objects by orienting a probe attached at each point. In addition, several opacity combinations for the line textures on the outer and inner volumes were investigated to determine which combination worked best. To do so, probes were placed onto the boundary of the irregular

volume objects. Since their boundaries were approximated by the quadrilateral tessellation of the irregular surfaces, this tessellation was used to guide the placement of the probes and to generate corresponding view parameters. The outer line texture opacity varied at four discrete levels, i.e. $0, 0.05, 0.1, 0.2$ and the inner line texture opacity at $0, 0.1, 0.2, 0.4$. Opacity 0 was included to allow the no-texture case to be considered. The opacity of the inner line texture was doubled because it is subject to visibility attenuation caused by the outer volume. Therefore, it needs a higher opacity to be reasonably visible. The experiments tested all opacity combinations of the outer and inner line textures, giving 16 combinations in total. For each, three probes were placed on the boundary of the outer volume object as well as on that of the inner, meaning there were 96 tasks in each experiment sequence for each participant.

Next, the probe positions and corresponding view parameters were generated. The view parameters of a probe give it a particular view perspective. In the experiments, the participants were not allowed to rotate the scene to look at the volume from different perspectives. For two reasons, they were required to look at the probe from a fixed view, randomly generated following certain guidelines. First, free view gives the participants extra information because they can use global information and logical inference to understand the visualization problem at hand. However, this interferes with the purpose of these experiments which was to tell whether performance is enhanced or reduced by the line texture. Second, free view makes it difficult to control the exact visual information that the participants utilize to understand the problem because it is impractical to track all user behaviors and then to quantify and analyze these behaviors statistically. Therefore, a set of view parameters for each probe was fixed but randomly generated to ensure statistical equivalence subject to the following guidelines:

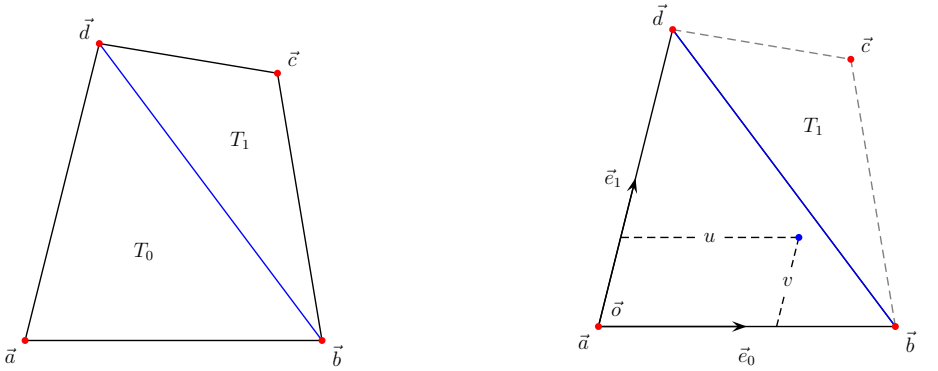
- The view position must be on the same side of the volume as the probe. While this requirement does not guarantee visibility of the probe, not adhering to it ensures invisibility.
- The view distance must be reasonably chosen so that in a perspective projection the volume objects cover the display window as much as possible without being clipped.
- The view position is within a 40° degree cone centered at the normal direction corresponding to the probe position to ensure that the participant has a reasonable view for understanding the normal direction.
- The probe position is within a 40° degree cone centered at the view direction and position to

ensure that the participant has a reasonable view of the probe because it is usually difficult to understand the scene from a glancing angle.

- The line segment from the probe position to the view position must not intersect the surface on which the probe is attached.

The display window, designed with 2400×2400 pixels, was centered on the physical screen. Because all the irregular shapes were built from spheres, their projections on the screen can be bounded more tightly by a square than by a rectangle, meaning that objects cover more of the window area in perspective projection. Therefore, a square display window was the best configuration and 2400×2400 was the largest square in pixels that can fit on our 3840×2400 resolution screens.

The probes were placed so that every point on the volume object boundary had an equal chance to be chosen to ensure both easy and difficult cases were included in the experiments. To achieve this, the quadrilateral tessellation of the irregular surface was considered as a sequence of N quadrilaterals, with each being assigned a unique index from 0 to $N-1$. A uniform integer random number in $[0, N-1]$ was then drawn to select one of the quadrilaterals. Next, an inner diagonal ($\overline{bd}$ in Figure 3.2.11a) which split the quadrilateral into two adjacent triangles T_0 and T_1 was randomly selected. The configuration is illustrated in Figure 3.2.11a. The areas of the triangles were then

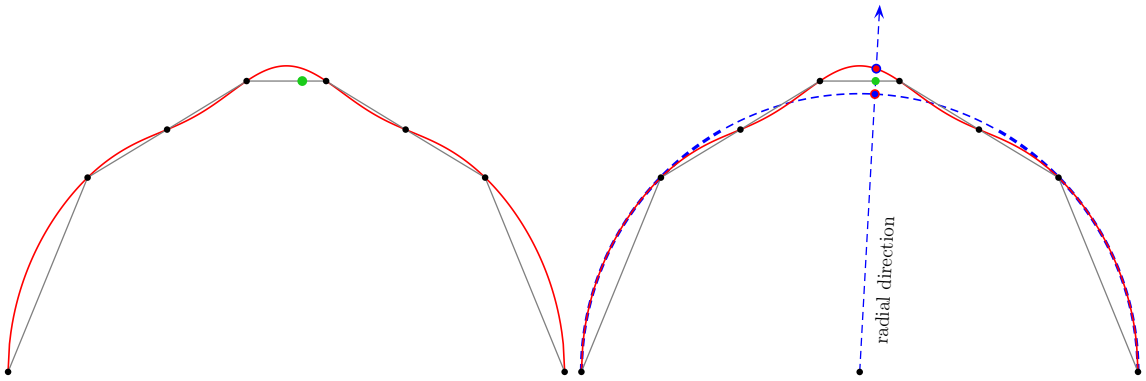


(a) Randomly Chosen Diagonal (b) Randomly Selected Point in T_0

Figure 3.2.11: Randomly Select a Point in Quadrilateral

computed as A_0 and A_1, and a critical threshold $t \in (0, 1)$ was formed as $A_0/(A_0 + A_1)$. A uniform random number $\tilde{t}$ was then drawn from $[0, 1]$ and compared with t to select either T_0 if $\tilde{t} \leq t$ or, if not T_1. The vertex of the triangle chosen that faced the splitting diagonal was considered as the origin $\vec{o}$, and its two connecting edges as axes $\vec{e}_0$ and $\vec{e}_1$, as illustrated in Figure 3.2.11b. This process

established a local frame on the triangle, one that could be non-orthogonal and non-uniform. A pair of uniform random numbers u and v in $[0, 1]$ were then drawn such that $u+v \leq 1$ and used to compute a random point on the triangle using $\vec{o} + u\vec{e}_0 + v\vec{e}_1$. This process, as illustrated in Figure 3.2.11b, resulted in a random point inside or on the boundary of the quadrilateral. This method worked for all quadrilaterals on the irregular surface regardless of convexity or whether the vertices were coplanar. The random point, however, did not necessarily satisfy the mathematical equation of the irregular shape exemplified by the green point in Figure 3.2.12a because the quadrilateral tessellation



(a) Random Point Not on Surface (b) Random Point Relocated on Surface

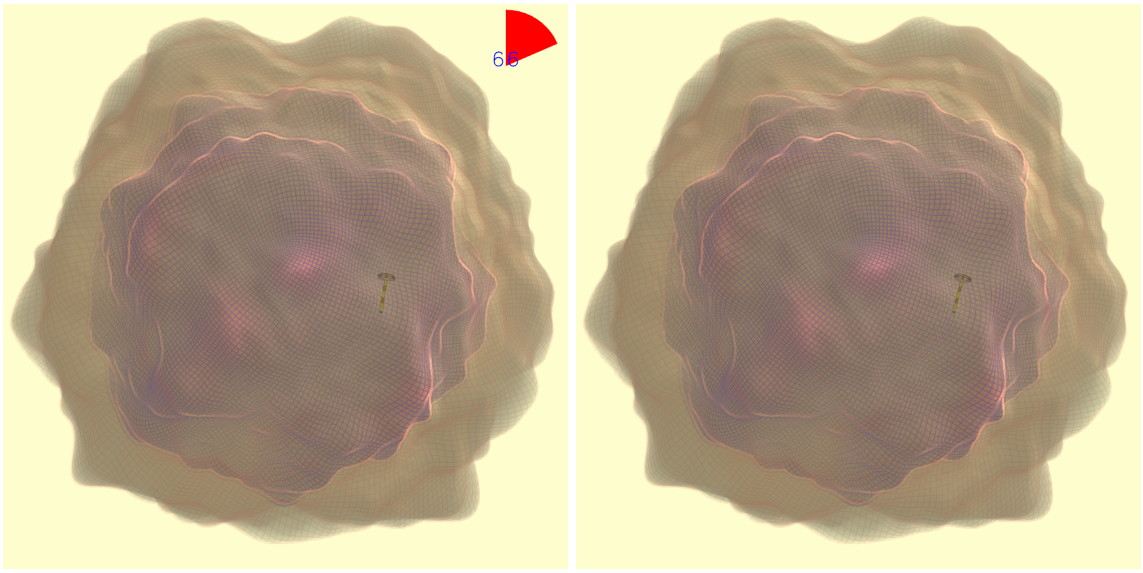
Figure 3.2.12: Relocate Random Point onto Surface

gave only an approximation of the irregular shape. However, this point identified a radial direction (blue dashed arrow line in Figure 3.2.12b) when connected to the origin. The displacement caused by the Gabor functions in this radial direction can be computed and used to offset the sphere point (blue point in Figure 3.2.12b) identified by the radial direction. The displaced sphere point (red point in Figure 3.2.12b) was then guaranteed to be on the irregular shape because of the way the irregular shape was constructed. This process is illustrated in Figure 3.2.12b. This method worked regardless if the final placement position is in a convex or a concave region of the surface.

3.2.6 Generation of Experiment Sequence

An experiment sequence was generated for each participant. First, irregular shapes were generated following the steps described in Section 3.2.2. Their corresponding subdivision surfaces, line textures and volume data were then generated as described in Section 3.2.3 and 3.2.4. In the third step, random probe positions and corresponding view parameters were generated based on Section 3.2.5. There were 96 probes in total. For the one volume object case, they were all placed

on the same object; for the two object case, they were placed equally on both. The last step was repeated once to generate probes for training purposes. Before an experiment sequence started, the participant was provided with a training phase in which they oriented the probe in exactly the same way as in the experiments, the only difference was that an error panel was displayed to show in degrees how far the current probe direction was from the normal direction computed. This training familiarized the participant with the system. Figure 3.2.13 compared the training and the experiment phases. When the participant was ready, the experiment sequence was started. The entire sequence



(a) Training (b) Experiment

Figure 3.2.13: Two Phase User Study

was conducted without breaks. For each presentation in the sequence, the participant was allowed sufficient time as he or she needed. A satisfactory probe orientation was indicated by right clicking the mouse. On detecting the click, the system began to render the next presentation and displayed the error panel for 2 seconds, showing the degree difference between the estimated and the computed normal directions for the presentation just finished. The system also recorded other information, including the time spent, the probe direction and position, the estimated normal direction and the actual normal direction.

Two experiment sequences were run for each participant, with one involving only one opaque volume object and the other two volume objects. The participants were asked to wait at least one day between the two sequences to ensure their performance was not affected by fatigue. The

participants included 11 volunteers, both faculty members and graduate students, all with computer graphics backgrounds and/or relevant coursework to ensure they understood the tasks correctly. All participants finished as expected except for two who were unable to finish both experiment sequences due to scheduling conflicts. Thus, the resulting data for analysis included 1056 records for the two volume object case and 864 records for the one volume object case.

3.3 Perceptual Validation of Projected Textures

The data were analyzed in several ways depending on how the normals were computed and how the collected information was utilized. The primary interest was to see if the line texture added helped humans to perceive the shape of the objects better. Since the quantity used to measure the performance of perception was the angular error between the estimated normals and the actual normals, significant difference between the angular error from various cases was looked for. There was one case for each line texture opacity, including the case for no texture (0 opacity). The statistical analyses and results are presented below, with supporting tools from the R package [71].

Intuitively, the normal can be computed based on Equation 3.2.4 as it gives an exact definition of the gradient vector at any point on the irregular shape and the normal is known to be parallel to the gradient vector, subject only to an orientation test that can be easily determined by the irregular shape design. However, comparing user estimates with normals computed from volume data considered the information loss introduced by visualization, as it was based on volume rendering of the volume data generated, and the participants' understanding of the objects completely relied on the visualization presented to them. The volume data, however, did not represent the original signal completely accurately as explained in Section 3.2.4. Moreover, even if the Nyquist theorem provides a theoretical guarantee of faithful reconstruction of the original signal provided sampling frequency is above the Nyquist frequency, the required reconstruction kernel to fully reach the theoretical potential demands infinite support. For efficiency, the volume renderer used here did not use such a kernel function, suggesting that the visualization did not fully utilize the information provided by the volume data, further implying that the participants' understanding and their corresponding estimation were affected by this approximation problem in the visualization process. Therefore, one of the limitations of this study is its ability to compare the estimated normal directions based on only an approximated version of the signal with the true normal directions derived from the complete and exact mathematical definition. As a result, we analyzed angular error in probe orientation against normals computed both analytically and by determining the gradient vector from the volume data through central difference and trilinear interpolation.

A different consideration requires a filtering operation applied on the data before analysis. The concern is that the perspective projection in the rendering caused some of the 3D information in the view direction to disappear. For 3D line elements, this degree of loss depends on the degree of

parallelism between the line direction and the view direction. When the line textures curve almost parallel to the view direction, they lose most of the curvature information they carry, making them almost useless in terms of conveying the shape of the covered object. To account for this in the data analysis, we filtered out all probes that fell into the central region of the display window, which is where the line texture loses most of its ability to convey shape. This loss of effect occurred because the base shape was a sphere, and in the center of the window, the sphere looked almost flat and any bumps added onto it pointed towards or away from the viewer. This problem was less serious where the shape curved away from the view direction, a situation occurring less when moving to the periphery of the view window. The display window was considered 2×2 in size with the center at $(0, 0)$, so any points within a 0.5 radius from the center were considered to be in the central area.

For the one volume object case, a one-way, within subjects ANOVA test was performed on the angular error from the four levels of opacity. When the normals were computed from the analytical gradient equations (Equation 3.2.4), the results were significant, with $F(3, 24) = 6.986$ and $p = 0.0015 < 0.05$. The mean angular errors in degrees and standard deviations for each level of opacity are listed in Table 3.3.1 and their box plot shown in Figure 3.3.1a. A pairwise t test revealed that the mean angular errors for opacity 0.05 and 0.2 were significantly less than that for opacity 0.

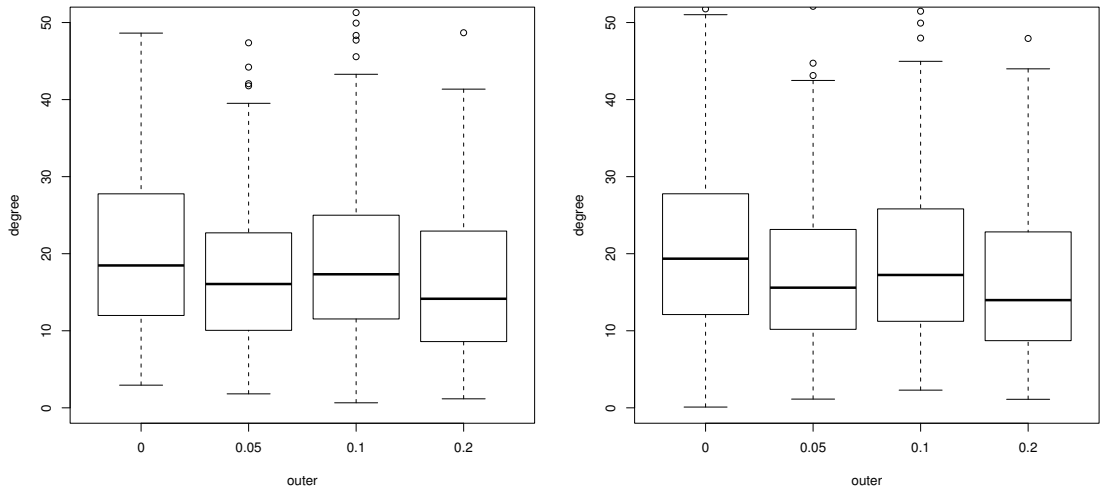
Table 3.3.1: All Data and Analytical Normal ($p = 0.0015$)

opacity	0	0.05	0.1	0.2
mean	21.0351	17.5596	19.0229	16.3117
std	12.1744	10.4626	11.1855	9.5994

When the normals were computed from the volume data based on central difference and trilinear interpolation, the results were significant, with $F(3, 24) = 8.093$ and $p = 0.0007 < 0.05$. The mean degree errors and standard deviations for each level of opacity are listed in Table 3.3.2 and their box plot shown in Figure 3.3.1b. A pairwise t test revealed that the mean degree errors for opacity 0.05 and 0.2 were significantly less than that for opacity 0.

Table 3.3.2: All Data and Interpolated Normal ($p = 0.0007$)

opacity	0	0.05	0.1	0.2
mean	21.4912	17.6193	19.3252	16.4327
std	12.1627	10.6690	11.3729	10.1045



(a) All Data and Analytical Normal ($p = 0.0015$) (b) All Data and Interpolated Normal ($p = 0.0006$)

Figure 3.3.1: Box Plot for One Volume Object (Generated by R [71])

The box plots in Figure 3.3.1 require further explanation. The bold horizontal black line marks the position of the median of the dataset, and the bottom and top edges of the box are located at the 25% and 75% quantiles of the dataset, referred to as the first and the third quantiles, respectively. The Inner Quantile Range (IQR) is the difference between the third quantile and the first quantile, i.e., the vertical length of the box. The outliers are identified as being less than the first quantile (bottom edge) minus 1.5 $\times$ IQR or greater than the third quantile (top edge) plus 1.5 $\times$ IQR, indicated by the empty circles in the box plot. The upper and lower short horizontal end lines are positioned respectively at the largest and smallest non-outliers in the dataset.

For the two overlapping volume objects case, the opacities of both the inner and outer line textures were factors. The four levels of each gave a total number of 16 combinations, which were assigned unique case IDs and labeled from 0 to 15 for ease of analysis. A one-way, within subjects ANOVA test was performed on the degree difference from the 16 combinations, the results showing no significant differences between having or not having line texture on any of the inner or outer volume objects.

When the filter was applied and the normals were computed from the volume data using central difference and trilinear interpolation, the results were significant, with $F(15, 147) = 2.093$ and $p = 0.0131 < 0.05$. The mean degree errors and standard deviations for each opacity combination are

listed in Table 3.3.3 and 3.3.4, respectively. The column highlighted in red enumerates the opacity levels of the outer line texture, and the row in blue lists that of the inner line texture. Each cell represents an opacity combination where the outer opacity is given by its row and the inner opacity is given by its column. The index in the parentheses is the case ID corresponding to the opacity combination. Figure 3.3.2 shows the box plot of the data. A subsequent t test revealed only that the mean degree error for Case 7 (green cell in Table 3.3.3 and 3.3.4) was significantly less than that of Case 0, which corresponds to the base case where no line texture was used.

Table 3.3.3: Mean Degree Error of Filtered Data and Interpolated Normal ($p = 0.0131$)

	0	0.1	0.2	0.4
0	21.9622 (0)	21.0777 (1)	23.6670 (2)	21.7862 (3)
0.05	25.6061 (4)	20.4664 (5)	28.5371 (6)	17.8850 (7)
0.1	20.2023 (8)	22.7455 (9)	18.8277(10)	21.6268(11)
0.2	20.3478(12)	19.8844(13)	19.2951(14)	18.2464(15)

Table 3.3.4: Standard Deviation of Filtered Data and Interpolated Normal ($p = 0.0131$)

	0	0.1	0.2	0.4
0	11.02834 (0)	12.14536 (1)	15.11749 (2)	12.3579 (3)
0.05	15.35958 (4)	8.66895 (5)	8.66206 (6)	11.4138 (7)
0.1	9.46719 (8)	13.18576 (9)	10.07171(10)	14.7190(11)
0.2	10.07273(12)	13.35568(13)	10.34285(14)	10.3749(15)

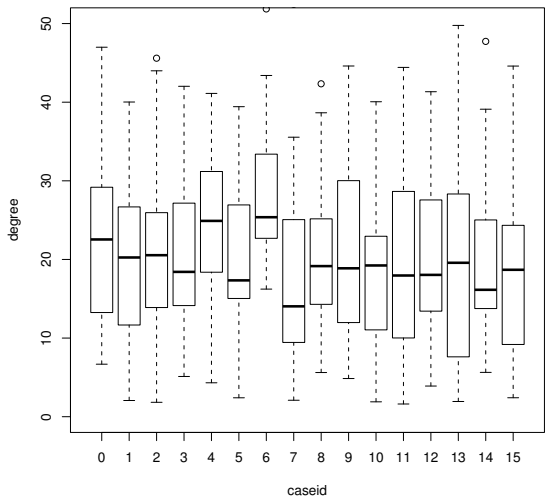


Figure 3.3.2: Box Plot for Filtered Data and Interpolated Normal ($p = 0.0131$)

A further analysis considered the attached surface of the probe. Since the probe could be

attached to the outer or inner volume object, this additional factor gave a total number of 32 cases, uniquely numbered from 0 to 31 as listed in Table 3.3.5. When the filter was applied and the normals computed from the volume data using central difference and trilinear interpolation, the results of an ANOVA test were significant, with $F(31, 249) = 1.519$ and $p = 0.0442 < 0.05$. Figure 3.3.3 shows the box plot of all 32 cases. A subsequent t test revealed that the mean degree errors for Cases 17, 23, 25, 26, 27, 30 (green in Table 3.3.5) were all significantly less than that of Case 16 (dark gray in Table 3.3.5), which corresponds to the base case where no line texture was used.

Table 3.3.5: Correspondence Between Case ID and Factor Combinations

Case ID	Outer Opacity	Inner Opacity	Probe Attached Object
0	0.0	0.0	outer
1	0.0	0.1	outer
2	0.0	0.2	outer
3	0.0	0.4	outer
4	0.05	0.0	outer
5	0.05	0.1	outer
6	0.05	0.2	outer
7	0.05	0.4	outer
8	0.1	0.0	outer
9	0.1	0.1	outer
10	0.1	0.2	outer
11	0.1	0.4	outer
12	0.2	0.0	outer
13	0.2	0.1	outer
14	0.2	0.2	outer
15	0.2	0.4	outer
16	0.0	0.0	inner
17	0.0	0.1	inner
18	0.0	0.2	inner
19	0.0	0.4	inner
20	0.05	0.0	inner
21	0.05	0.1	inner
22	0.05	0.2	inner
23	0.05	0.4	inner
24	0.1	0.0	inner
25	0.1	0.1	inner
26	0.1	0.2	inner
27	0.1	0.4	inner
28	0.2	0.0	inner
29	0.2	0.1	inner
30	0.2	0.2	inner
31	0.2	0.4	inner

The difference between the 16-case and the 32-case analyses showed that the added line texture was more effective on the inner object, which makes intuitive sense because it was enclosed

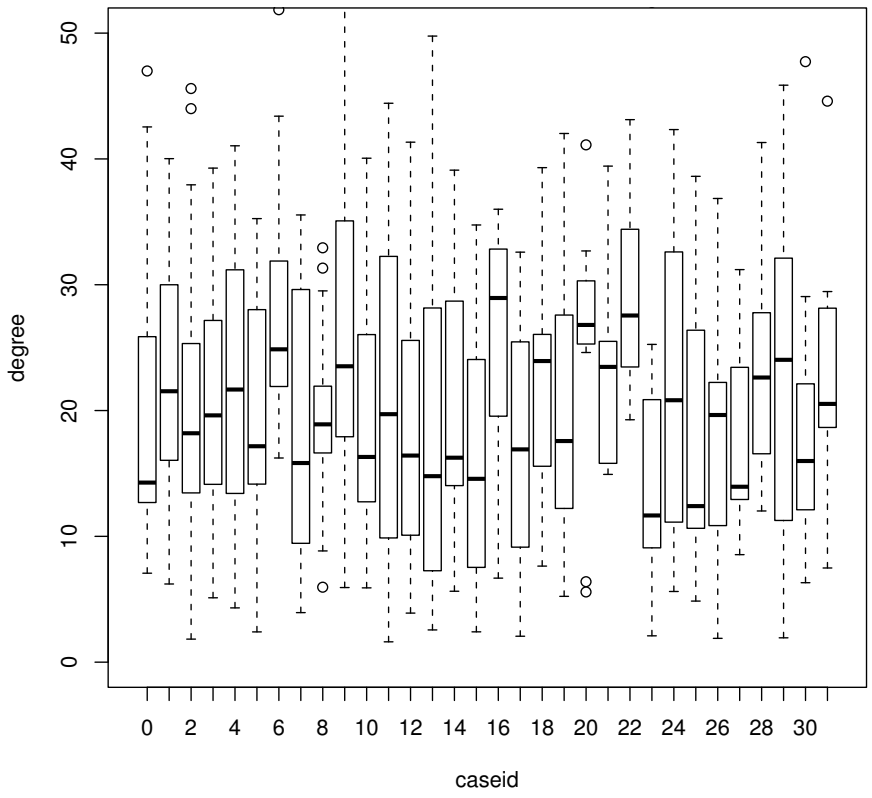


Figure 3.3.3: Filtered Data and Interpolated Normal for 32 Cases ($p = 0.0442$)

by the outer volume object spatially, thus decreasing its visibility and making it difficult to perceive. In this case, the line texture gave additional visual cues to aid in shape perception.

This chapter covered the implementation of the volume renderer used in this research including several factors beyond the theoretical issues discussed in Chapter 2.2 and the corresponding choices made. It also discussed the user study conducted to verify the effectiveness of auxiliary line textures, including the details explaining the design of the irregular shapes used, the method for generating the required grid-like line textures for these shapes, the approach for producing the volume datasets used in rendering, the parameters for the experiments including probe placements and line texture opacities studied and the generation of experiment sequences. The data collected showed that the added line textures helped the perception of irregular shapes in volume rendered images. The results also revealed several issues, one of which being that the line texture produced

by the method introduced in this chapter did not work well near the center of the display window, a situation probably caused by not considering the intrinsic property of the shape when producing the line texture. The results of this study is then used to guide the following research.

Chapter 4

Surface Conforming Grid Textures

As discussed in Chapter 3, the data analysis of the results from the user study found that the grid-like line texture did not work very well in the central region of the screen because it lost the curvature information carried under the influence of perspective projection. Therefore, a better approach is needed to generate lines to address this concern. One solution is to use lines with directions conforming to the intrinsic geometrical properties of a surface, a suggestion that has been investigated in previous studies [36]. Section 2.3 introduced several types of lines defined by differential geometry. Of particular interest are those following the two principal curvature directions as they are guaranteed to be perpendicular everywhere the principal curvatures are not identical. For the complex shapes usually encountered in volume data, most points on the surface have unequal principal curvatures, allowing principal curvature direction guided line texture to be applicable.

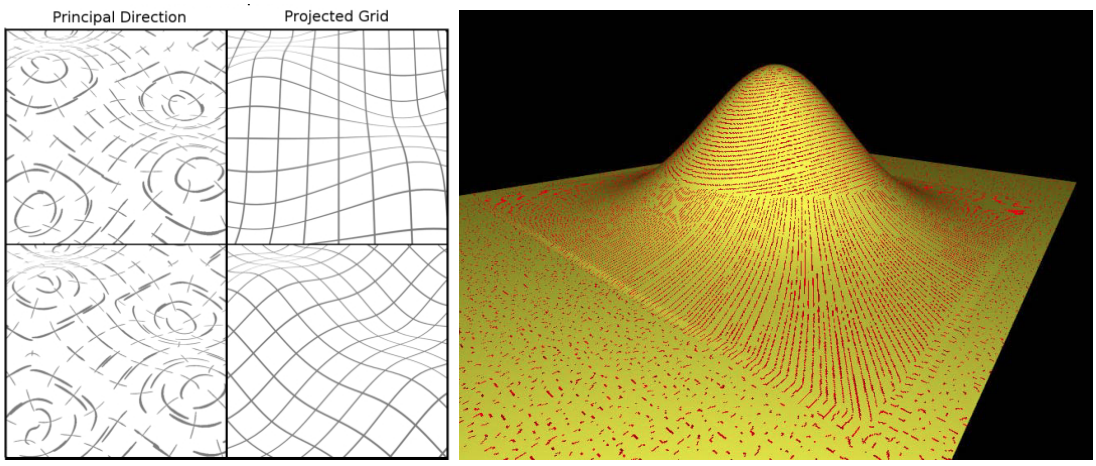
Principal curvature directions, however, have their problems. Therefore, we prefer a method that utilizes principal curvature directions while avoiding the disadvantages. In helping humans to better understand a visualization target, we would like to emphasize its large scale features since such features usually give the most important identifications to the objects. For example, a torso has many details on a close examination, but its general shape is a cylinder. Such a view demands a way to generate grid-like line textures to describe large scale structures, which in turn requires to identify them first. One quantitative way to do so is to evaluate the importance of different locations on a visualization target, realized, for example, through expressing importance by different scalar values. Then the principal curvature directions of these locations can be utilized to guide the generation of a grid structure of the entire object, a process realized in two steps. The first one generates a

globally consistent cross field for the visualization target based on the principal curvature directions of the important locations. This provides a set of vectors for each location on the object, identifying the important directions at the same place. The second step produces two sets of lines that follow the cross field as close as possible, thus inheriting the globally consistent behavior and providing an explicit representation needed by our rendering algorithm.

Since human perception is closely related to subjective understanding, we think it wise to allow humans control of the grid structure generated. For example, instead of using principal curvature directions, users should be able to manually provide guiding directions that they consider important. By the same logic, it is also beneficial to allow them to identify the important locations. Having this in mind, the following sections present detailed explanations of a method that we developed based on existing research, combined and adjusted for our specific need.

4.1 Principal Curvature Directions

As mentioned above, simply following principal curvature directions is not completely effective. Figure 4.1.1 shows principal directions without further manipulation. Figure 4.1.1a shows



(a) Two Principal Directions (b) First Principal Direction (Mahdizadeh [60])

Figure 4.1.1: Principal Directions

four views of the same geometry involving four Gaussian bumps, two indenting and two protruding. The geometry is covered by two types of line textures: the left column uses a line texture formed by sampling the surface and following streamlines from the sample points in principal curvature directions, while the right column uses projected grid texture, aligned with the viewing frame in the top row and rotated 45° in the bottom. Figure 4.1.1b uses an algorithm developed by Jobard and Lefer [39] to place evenly spaced streamlines following the first principal curvature direction on a Gaussian bump. Figure 4.1.1b reveals two problems using lines following principal curvature directions. First, the first principal direction abruptly changes its direction by 90° as streamlines proceed down the bump. Second, in the flat region at the base, the principal curvature direction is not unique, resulting in randomness in the streamlines produced. For our work, we need a line grid that is generally aligned with the two principal directions but that is uniform as much as possible except at a few singularities, as exemplified in Figure 4.1.2. Another problem with principal curvature directions is that they are easily affected by noisy data, presenting an incoherent structure of the constructed lines. Ideally, the line texture needs to capture only the high level important features of the underlying shapes without including unimportant features or noise in the volume data. Work on generating quadrilateral meshes for geometric models has shown that one successful approach is

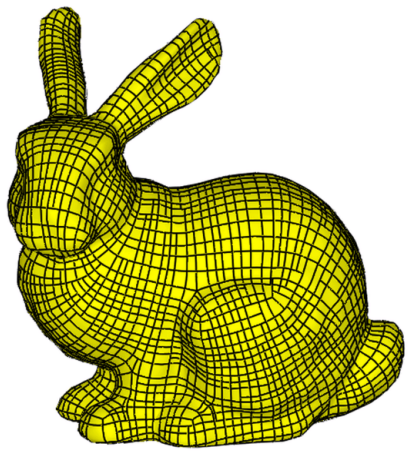


Figure 4.1.2: Line Texture Grid on Bunny (Palacios and Zhang [70])

to choose important places for evaluating the two principal curvature directions and spread their information to the entire shape of interest [70]. In the resulting derived direction field, the directions are dominated and guided by the principal directions of the places chosen. The problems in Figure 4.1.1 are addressed by assigning a few points with well-defined principal curvature directions to guide the entire texture. This method also has the advantage that it better suits general shapes, therefore addressing the disadvantage of the projective approach used so far, which works only for star-shaped surfaces with explicit representations.

In our target applications, since we are only interested in the shape of the objects being visualized, we can always extract these shapes as polygonal surfaces, for example, by running marching cubes with respect to a chosen isovalue. Such a polygonal surface representation makes it possible to apply existing surface based analysis techniques, including global parameterization methods, to construct a mapping between the surface and a 2D Cartesian grid. Having this mapping allows the extraction of two sets of orthogonal lines by sampling along the two axis directions in the parametric space. When sampled at a constant density, the sampling grid in the parametric space presents a grid-like structure on the original polygonal surface, thus providing the essential line elements that we need. Such a mapping might introduce metric or angular distortions, which manifest as varying quadrilateral sizes or non-orthogonal intersections between members from the two sets of lines. Depending on the inherent properties of a surface, it is not always possible to eliminate these distortions. Thus, most parameterization techniques strive to minimize them, which is usually captured in a constrained optimization process.

One such technique was presented in [7] by Bommes *et al.* It relies on a smooth cross field to guide the global parameterization of a 2-manifold. Their original approach involves two steps, with the first one producing the cross field and the second the parameterization. Both steps are efficiently solved by a publicly available greedy mixed integer solver [8] that they designed specifically for this class of problem. A cross field associates each point on the surface with a pair of orthogonal vectors that define the tangent plane at that point. These vectors, paired with their negations, form a cross, hence a local frame, at that point. For example, the principal curvature directions and their negative directions, when well defined, can be used as a cross field for this purpose, although this would have the problems discussed earlier, making it less suitable for our purpose. In particular, we would like to have control of the cross field generation, specifically the placement of singularities, which should be on locations representing important geometrical features.

To this end, the work done by Palacios and Zhang [70] fits well, as their technique propagates an N-way rotational symmetry (N-RoSy) field across an entire 2-manifold from specified initial locations where local N-RoSy elements are provided. The resulting field is smooth, with continuous and coherent variations, a desirable attribute for our target applications. Their work is general in the sense that the formulation is based on rotational symmetry, which includes a cross field as a special case for $N = 4$. The principal curvature directions at specifically chosen locations are used as initial conditions for propagation. This ensures the generated cross field is dominated by the chosen principal curvature directions but not constrained by the principal curvature direction field of the surface everywhere, an approach with an overall effect of emphasizing chosen features while minimizing less interesting locations. As Palacios and Zhang's work allows movement and cancellation of singularities, it is possible to rely on a manual process to generate and fine tune the cross field. However, automatic selection of initial locations is also possible by employing a further step to grasp salient features of a surface.

The idea is to define a scalar field on the surface that describes the importance of each surface point. Since the ultimate goal of applying the grid-lines is to help enhance the perception of shapes, saliency [38] is a good quantity to use as it captures visually important features. Lee *et al.* [56] extended the idea of saliency to 3D polygonal surfaces, where the saliency of a vertex is defined on a center-surround evaluation of its neighbors' features such as mean curvature. This evaluation is applied to several different scales with increasing radius. Each scale determines a saliency map for all of the vertices in the mesh, and the final saliency map is computed to be a non-linear sum of the

saliency maps at the chosen scales. Visually important positions then stand out as local extrema in the saliency map.

Saliency was originally used in 2D image analysis [38] where it was derived from the human visual search strategies explained by the feature integration theory [81]. A biologically-plausible architecture was then proposed by Koch and Ullman [48] on which several models, including saliency analysis, were based. The idea is that saliency-based visual analysis reflects how humans search for important features visually. Therefore, a saliency field of the surface encodes the importance of each point, with higher values indicating more significance.

As most of the involved techniques are based on triangle meshes with 2-manifold topology, the framework employed in our research expects such polygonal surfaces and it includes these steps with details presented in the following sections:

1. Extract a polygonal surface representing the object of interest, for example, by the Marching Cubes [59] algorithm.

2. Compute saliency for all vertices on the mesh and select the important ones.

3. Generate a smooth cross field based on selected initial locations and their principal curvature directions.

4. Parameterize the mesh based on the cross field.

4.2 Using Mesh Saliency to Locate Surface Critical Points

Mesh Saliency [56] was designed to encode the importance of each vertex of a polygonal mesh, based on an empirical study of human visual search strategy. As such, it grasps the features of a polygonal surface that visually appear more important to humans. The computation of mesh saliency is based on mean curvatures. There are several methods to compute the mean curvature such as the one developed by Taubin and Gabriel [80]. We use the method by Watanabe *et al.* [82] which is summarized below.

4.2.1 Mean Curvature

For any vertex $\vec{v}$ on a triangle mesh, let $\hat{n}$ be the unit normal vector at $\vec{v}$ and $\hat{t}$ be a unit vector in the tangent plane at $\vec{v}$ and let $\hat{t}_{\max}$ and $\hat{t}_{\min}$ be the two principal curvature directions. If ϕ is the angle formed by $\hat{t}$ and $\hat{t}_{\max}$, then according to Euler's formula

$$\kappa_{\hat{n}}(\phi) = \kappa_{\max}\cos^2\phi + \kappa_{\min}\sin^2\phi \tag{4.2.1}$$

where $\kappa_{\max}$ and $\kappa_{\min}$ are the principal curvatures at $\hat{t}_{\max}$ and $\hat{t}_{\min}$. The mean curvature H at $\vec{v}$ is given by

$$\frac{\kappa_{\max}+\kappa_{\min}}{2} = \frac{1}{2\pi}\int_0^{2\pi}\kappa_n(\phi)\mathrm{d}\phi. \tag{4.2.2}$$

To determine the integral on the right hand side, we consider $\vec{v}$ on a smooth surface and parameterize the smooth curve $\vec{x}(s)$, formed by the normal section and contained in the surface, by its arc-length s, where $\vec{x}(0)$ corresponds to $\vec{v}$. Therefore, we have $x'(0) = \hat{t}$ and $\vec{x}''(0) = \kappa_{\hat{t}}(\vec{v})\hat{n}$. Given such a construct, the Taylor expansion of $\vec{x}(s)$ to its second order around the vicinity of $\vec{v}$ is

$$\vec{x}(s) \approx \vec{x}(0) + s\vec{x}'(0) + \frac{s^2}{2}\vec{x}''(0) + O(s^3) = \vec{v} + \hat{t}s + \frac{1}{2}\kappa_{\hat{t}}(\vec{v})\hat{n}s^2 + O(s^3). \tag{4.2.3}$$

Taking the inner product with $\hat{n}$, and observing that $\hat{t}$ and $\hat{n}$ are orthogonal give

$$\kappa_{\hat{t}}(\vec{v}) = \lim_{s\to 0}\frac{2\langle\hat{n}, \vec{x}(s)-\vec{v}\rangle}{\|\vec{x}(s)-\vec{v}\|^2} \tag{4.2.4}$$

where $\|\vec{x}(s) - \vec{v}\| \approx s$ for small s. In a discrete setting such as on a triangle mesh, the normal curvature $\kappa_{\hat{t}}(\vec{v})$ at direction $\hat{t}$ can be approximated as

$$\kappa_{\hat{t}}(\vec{v}_i) \approx \frac{2\langle \hat{n}, \vec{v}_i - \vec{v} \rangle}{\|\vec{v}_i - \vec{v}\|^2} \tag{4.2.5}$$

where $\vec{v}_i$ is incident to $\vec{v}$ through an edge on the polygonal surface and $\hat{t}$ is the direction of $\vec{v}_i - \vec{v}$ projected into the tangent plane.

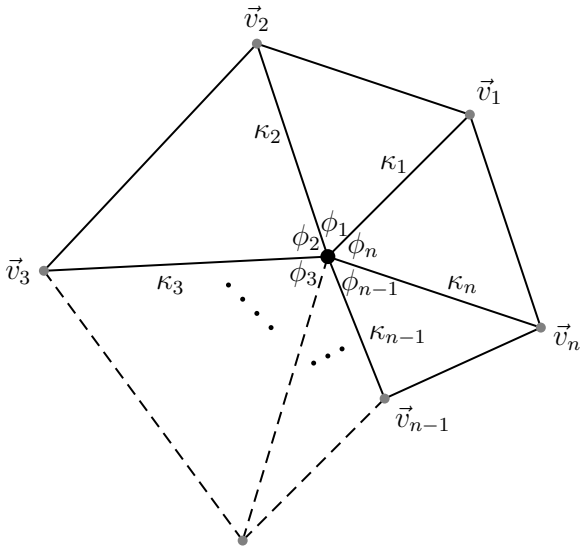


Figure 4.2.1: Mean Curvature Approximation

Therefore, the mean curvature as expressed in Equation 4.2.2 is computed as a Riemann sum along the first ring of vertex $\vec{v}$ as

$$H \approx \frac{1}{2\pi}\left[\kappa_1\left(\frac{\phi_n + \phi_1}{2}\right) + \kappa_2\left(\frac{\phi_1 + \phi_2}{2}\right) + \cdots + \kappa_n\left(\frac{\phi_{n-1} + \phi_n}{2}\right)\right] \tag{4.2.6}$$

where n is the number of incident vertices of vertex $\vec{v}$ and κ_i is the curvature along the i^{th} edge from $\vec{v}$ to $\vec{v}_i$ and ϕ_i is the angle formed by the i^{th} and $i + 1^{st}$ edges at $\vec{v}$ as illustrated in Figure 4.2.1. Equation 4.2.6 is effectively a midpoint scheme.

4.2.2 Saliency

The mean curvature creates a function $H : R^3 \mapsto R$ that maps each vertex of a polygonal mesh to its mean curvature. For any vertex $\vec{v}$, to compute the saliency $S(\vec{v})$ based on $H(\vec{v})$ requires

considering the Gaussian average of all $H(\vec{v}_i)$ where $\vec{v}_i$ is within a neighborhood of $\vec{v}$. The neighborhood of $\vec{v}$ is the set of mesh vertices whose distance to $\vec{v}$ is less than a given radius for some definition of distance, such as the geodesic distance. In our work, we have used the Euclidean distance of the embedding space to define the neighborhood. Therefore, the neighborhood $N(\vec{v}, \sigma)$ of vertex $\vec{v}$ on a polygonal mesh $\mathcal{M}$ is the set of vertices

$$N(\vec{v}, \sigma) = \{\vec{x} : \|\vec{x} - \vec{v}\| < \sigma \wedge \vec{x} \in \mathcal{M}\}. \tag{4.2.7}$$

The Gaussian average of the mesh curvature for a given neighborhood $N(\vec{v}, \sigma)$ of vertex $\vec{v}$ is then defined as

$$G\left(H(\vec{v}), \sigma\right) = \frac{\sum\limits_{\vec{x} \in N(\vec{v}, 2\sigma)} H(\vec{x}) \mathrm{e}^{-\frac{\|\vec{x} - \vec{v}\|^2}{2\sigma^2}}}{\sum\limits_{\vec{x} \in N(\vec{v}, 2\sigma)} \mathrm{e}^{-\frac{\|\vec{x} - \vec{v}\|^2}{2\sigma^2}}} \tag{4.2.8}$$

assuming a 2σ cut-off distance for the Gaussian kernel. The saliency $S(\vec{v})$ of vertex $\vec{v}$ at scale σ is formed as

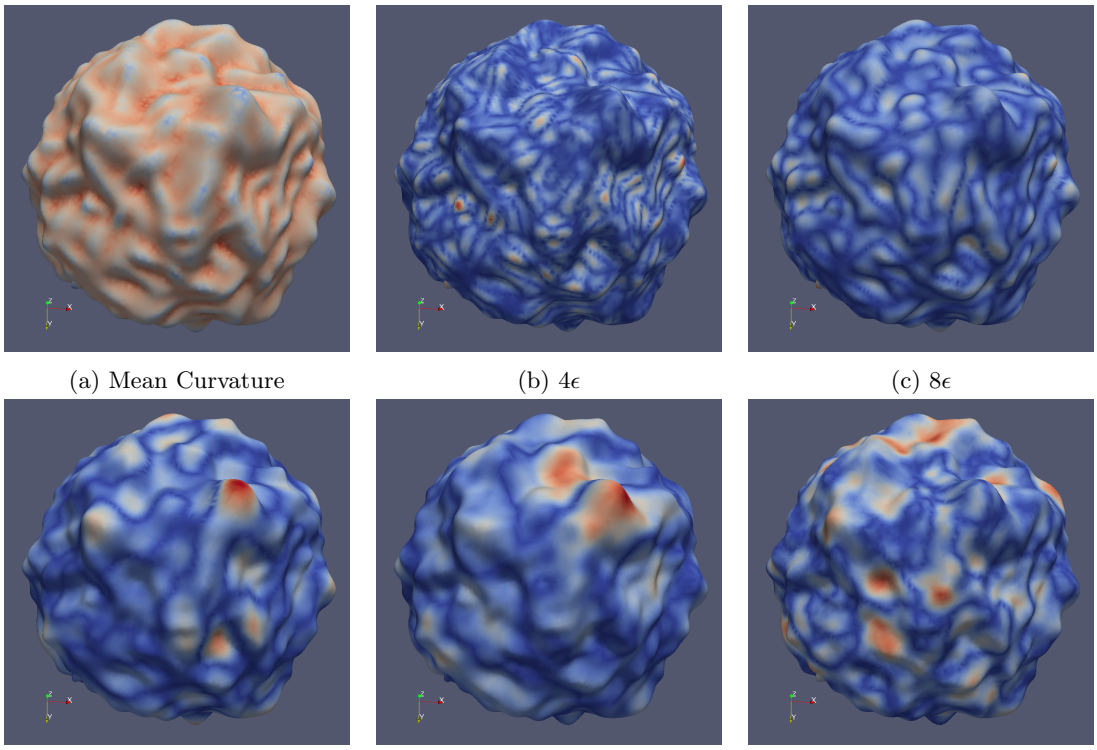
$$S(\vec{v}, \sigma) = \|G(H(\vec{v}), \sigma) - G(H(\vec{v}), 2\sigma)\|. \tag{4.2.9}$$

For each vertex of the polygonal mesh, saliency is computed at five different scales $4\epsilon, 8\epsilon, 16\epsilon, 32\epsilon, 64\epsilon$ respectively, where ϵ is 0.3% of the diagonal length of the model bounding box. Figure 4.2.2 illustrates the mean curvature and saliency values at different scales for a gabor mesh. The color coding assigns warmer color to higher values. These figures demonstrate the ability of mesh saliency to capture visually salient features at different scales corresponding to the σ parameter in Equation 4.2.9. For example, Figure 4.2.2b displays several small scale features while Figure 4.2.2e only highlights a few large ones. Therefore, features at different scales are captured separately and then combined to give an overall rating of their importance, a process simulating humans visual search strategy.

Specifically, for each vertex, the saliency values at five different scales are combined through a non-linear suppression operator as proposed by Itti [38], which helps to suppress unimportant features. This is achieved by the following steps done for each of the five saliency scales:

1. Linearly normalize the saliency values to $[0, 1]$, which provides a common foundation for further combination across different scales.

2. Compute the global maximum M and the average m of all local maxima m_i excluding the



(a) Mean Curvature (b) 4ϵ (c) 8ϵ

(d) 16ϵ (e) 32ϵ (f) 64ϵ

Figure 4.2.2: Saliency Value at Different Scales

global maximum.

3. Scale all saliency values by $(M - m)^2$.

At last, the mesh saliency of a vertex is computed as the summation of all saliency values at five different scales computed for that vertex as described above. For the same gabor surface show in Figure 4.2.2, Figure 4.2.3 shows the color coding of the mesh saliency.

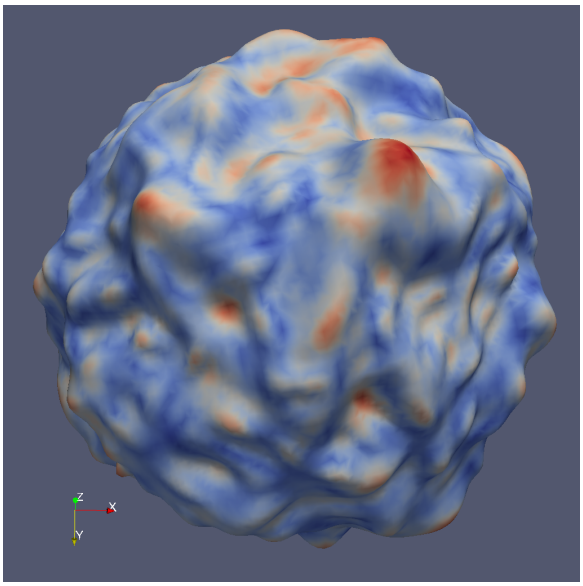


Figure 4.2.3: Mesh Saliency

4.3 Generating a 4-Way Rotational Symmetry Field

After using mesh saliency to identify the location of important geometrical features, the principal curvature directions at these locations are computed and used as guiding frames. The next step is to spread their direction information across the entire surface. The method reported by Palacios and Zhang [70] is used for this purpose. Their method is based on N-way rotational symmetry (N-RoSy), which intuitively captures structure invariant under rotations that are integer multiples of $\frac{2\pi}{N}$. The grid-like line texture, for example, is a specific case where $N = 4$ since the crossing feature gives 90° rotational symmetry. Figure 4.3.1 shows an example of a drawing whose hatch marks were guided by a 4-RoSy field covering the geometrical model. Mathematically, an N-RoSy field associates each spatial position with the set

$$\mathbb{S} = \left\{ \begin{bmatrix} r\cos(\theta + \frac{2k\pi}{N}) \\ r\sin(\theta + \frac{2k\pi}{N}) \end{bmatrix} : 0 \leq k \leq N-1 \right\} \tag{4.3.1}$$

of vectors that are $\frac{2\pi}{N}$ separated from each other. The members of $\mathbb{S}$ form a regular radial subdivision of a circle of radius r.

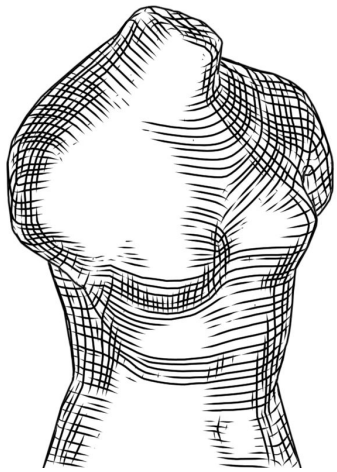


Figure 4.3.1: Drawing Made from a 4-RoSy Field on a Surface (Palacios and Zhang [70])

4.3.1 Representation Vector

An important element of the approach is the representation vector, which, for an N-RoSy $\mathbb{S}$, is defined as

$$\vec{\gamma}(\mathbb{S}) = \begin{bmatrix} r\cos(N\theta) \\ r\sin(N\theta) \end{bmatrix}. \tag{4.3.2}$$

The representation vector encapsulates the necessary information to fully recover the N-RoSy $\mathbb{S}$ and is independent of the choice of member vectors, because

$$N(\theta + \frac{2k\pi}{N}) \equiv N\theta \mod 2\pi \tag{4.3.3}$$

for any $k \in \mathbb{Z}$. The fundamental advantage of the representation vector is that it supports such common mathematical operations as addition, scalar multiplication and interpolation, all of which are difficult to form for Equation 4.3.1, due to the directional ambiguity. Specifically, N-RoSy addition and scalar multiplication are defined through representation vectors as

$$\begin{aligned} \mathbb{S}_1 + \mathbb{S}_2 &= \vec{\gamma}^{-1}\left(\vec{\gamma}(\mathbb{S}_1) + \vec{\gamma}(\mathbb{S}_2)\right) \\ \lambda\mathbb{S} &= \vec{\gamma}^{-1}\left(\lambda\vec{\gamma}(\mathbb{S})\right) \end{aligned} \tag{4.3.4}$$

where $\mathbb{S}, \mathbb{S}_1$ and $\mathbb{S}_2$ are N-RoSy elements, $\vec{\gamma}^{-1}$ is the inverse mapping of $\vec{\gamma}$ and λ is a real number. This property of the representation vector is important since well-defined mathematical operations

are needed for spreading the principal curvature direction information from seed locations, called design elements, to other places on the shape. A second advantage is that the representation vector allows methods from vector field analysis to be intuitively transferred to the analysis of N-RoSy fields. A representation vector, however, behaves differently under a change of coordinate system. In particular, a rotation

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{4.3.5}$$

applied onto a representation vector of an N-RoSy must take the adjusted form

$$R_\gamma = \begin{bmatrix} \cos(N\theta) & -\sin(N\theta) \\ \sin(N\theta) & \cos(N\theta) \end{bmatrix} \tag{4.3.6}$$

because the angle variation applied by the rotation is subject to further influence by the rotational symmetry, a fact that can be intuitively understood from Equation 4.3.3.

4.3.2 Singularities and Separatrices

Based on vector field analysis, a singularity of the representation vector field is a spatial position having a zero valued representation vector. As a result, it is impossible to recover the corresponding RoSy element following previous definitions. Such a spatial position is specially defined as a singularity of the corresponding RoSy field with an associated RoSy element whose member vectors are zeros as well. This establishment ensures an RoSy field share the same set of singularities with its corresponding representation vector field, thus making it convenient to study the RoSy field by examining the behavior of the representation vector field. Singularities may appear isolated or in cluster. An isolated singularity possesses an open neighborhood in which it is the only singularity. Our research involves RoSy fields that only possess isolated singularities. For a vector field defined on a 2-manifold, a singularity can either be a source, a sink, or a saddle. Singularities are important because they mark the features of a vector field, as well as quadrangulation for the underlying mesh.

A separatrix in a vector field is curve such that the vector defined at each point on the curve is identical to the tangent direction of the curve at the same point. Separatrices form the boundary between regions of a vector field that demonstrate different flow behavior.

4.3.3 Laplace Equation

Palacios and Zhang's work gives user control of singularities of an N-RoSy field, which they refer to as the design elements. In our work, the singularities correspond to the local extrema and saddles of the shape, therefore, the design elements are placed at these locations. Then the representation vectors at these places are computed and a Laplace equation $\nabla^2\gamma = 0$ is solved, with boundary conditions determined by the design elements to spread the field smoothly across all mesh vertices. The Laplace equation solution is based on the discrete form

$$\gamma_i = \sum_{\vec{v}_j \in N(\vec{v}_i)} \omega_{ij} T_{ij}(\gamma_j) \tag{4.3.7}$$

where γ_i, γ_j are the representation vectors at vertices $\vec{v}_i$ and $\vec{v}_j$ respectively, $N(\vec{v}_i)$ is the set of vertices incident to $\vec{v}_i$, ω_{ij} is the mean value coordinate and T_{ij} is the transport function from the tangent plane at vertex $\vec{v}_j$ to vertex $\vec{v}_i$.

4.3.3.1 Mean Value Coordinates

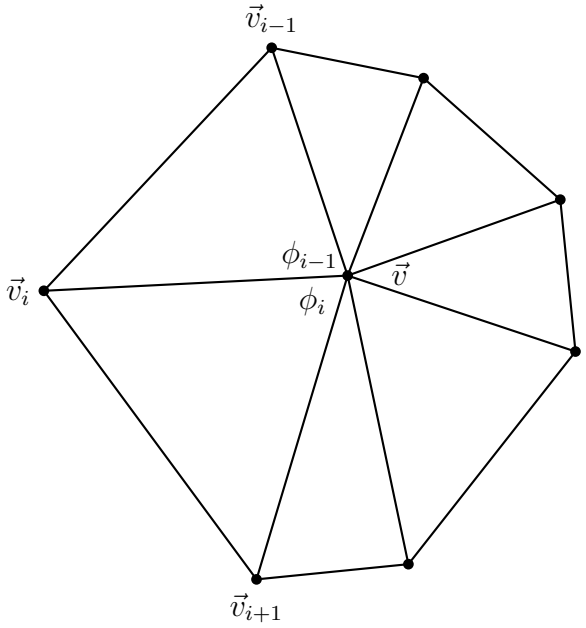


Figure 4.3.2: Mean Value Coordinates

The mean value coordinates [26] of $\vec{v}$ with respect to a group of n vertices $\vec{v}_1, \cdots, \vec{v}_n$ oriented counterclockwise, as illustrated in Figure 4.3.2 forming a star-shaped planar triangulation with $\vec{v}$, is

defined as

$$\lambda_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \quad \text{and} \quad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\vec{v}_i - \vec{v}\|} \tag{4.3.8}$$

where λ_i is the coordinate of $\vec{v}$ with respect to $\vec{v}_i$ for any $i \in \{1, \cdots, n\}$, and $\vec{v}_i$ is the vertex position. The mean value coordinates satisfies

$$\sum_{i=1}^{n} \lambda_i \vec{v}_i = \vec{v}, \quad \text{and} \quad \sum_{i=1}^{n} \lambda_i = 1 \tag{4.3.9}$$

and it is smooth and guaranteed to be positive.

4.3.3.2 Transport Function

The transport function T_{ij} in Equation 4.3.7 is the transformation from the tangent plane at vertex $\vec{v}_j$ to that at vertex $\vec{v}_i$ that maintains equivalent representation vectors. Let $\vec{\Gamma} : [0, 1] \mapsto \mathcal{M}$ be the surface geodesic from $\vec{v}_i$ to $\vec{v}_j$ such that $\vec{\Gamma}(0) = \vec{v}_i$ and $\vec{\Gamma}(1) = \vec{v}_j$, then two vectors $\vec{d_i}$ and $\vec{d_j}$ located at $\vec{v}_i$ and $\vec{v}_j$ respectively are equivalent if and only if

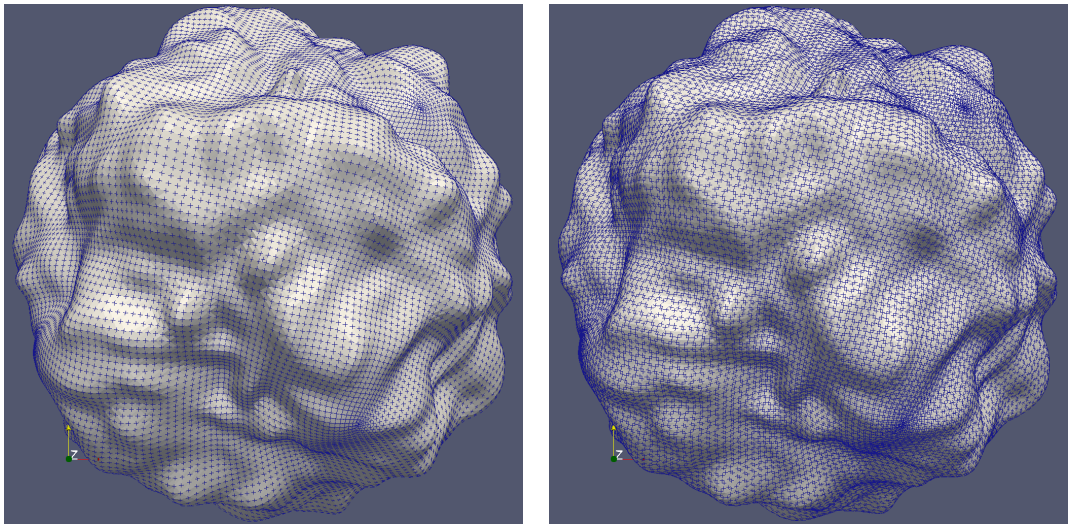
$$\alpha\left(\vec{\Gamma}'(\vec{v}_i), \vec{d_i}\right) = \alpha\left(\vec{\Gamma}'(\vec{v}_j), \vec{d_j}\right) \tag{4.3.10}$$

where $\alpha(\cdot, \cdot)$ forms the angle difference for its two vector operands. On a triangle mesh, the shortest path between two incident vertices is the edge that connects them. As such, an edge serves as the geodesic between its incident vertices. Therefore, the 2D rotation $R(\theta_{ij})$ maps d_j to its equivalent d_i, where $\theta_{ij} = \theta_i - \theta_j$, θ_i and θ_j are the angles from the x-axis to the geodesic at $\vec{v}_i$ and $\vec{v}_j$ respectively. Since a representation vector changes differently from a regular vector under rotational transformation as demonstrated in Equation 4.3.6, the transport function T_{ij} is computed as

$$T_{ij} = \begin{bmatrix} \cos(N\theta_{ij}) & -\sin(N\theta_{ij}) \\ \sin(N\theta_{ij}) & \cos(N\theta_{ij}) \end{bmatrix}. \tag{4.3.11}$$

Therefore, Equation 4.3.7 is a sparse linear system that can be solved efficiently with bi-conjugate gradient solvers. The solution is a divergence free vector field that defines a representation vector for every vertex. Well-defined mathematical operations supported by the representation vector field are essential in solving the Laplace equation; otherwise, the solution would likely produce undesirable results. Having the representation vector field allows recovery of a 4-RoSy field, as

illustrated in Figure 4.3.3, defined on each mesh vertex or the center of each mesh triangle. This provides the foundation for a global parameterization for the entire mesh.



(a) Per-Vertex 4-RoSy (b) Per-Face 4-RoSy

Figure 4.3.3: Gabor 4-RoSy Field

4.4 Generating a Surface Parameterization Supporting Grid-like Texturing

After a globally consistent cross field is established on the triangle mesh, the next step is to generate a parameterization that maps each vertex on the mesh to a point in a 2D Cartesian parametric space. In creating this parameterization, the goal is to minimize the least square error between the cross field and the gradient vectors of the two parametric directions, with the intention to minimize mapping distortion under the least square measure, a property that is highly desirable to generate grid-like line textures.

Mesh parameterization has been the focus of active research for a long time with fruitful results [44, 32]. Of the various approaches, perhaps the most successful, to date, in producing a uniform quadrangulation is that by Bommes *et al.* [7]. In their approach, the parameterization is formulated as a mixed integer optimization problem that guarantees a pure quadrangulation. Their technique includes an initial step, aiming to generate a smooth cross field. Instead of using this field, we elected to substitute our rotational symmetry field. This is preferred, since it is based on mesh saliency, and also allows manual editing. The second step is an optimization minimizing the local orientation energy defined as

$$E_T = \|h\nabla u_T - \hat{u}_T\|^2 + \|h\nabla v_T - \hat{v}_T\|^2 \tag{4.4.1}$$

where (u, v) are the coordinates of mesh vertices in the parametric space and the $\nabla(\cdot)_T$ operator forms the gradient vector for its operand with respect to triangle T. $\hat{u}_T$ and $\hat{v}_T$ are two orthogonal unit directional vectors extracted from the cross field at the center of triangle T, because the energy term E_T is formed as a per-triangle quantity. The scalar parameter h is user tunable, and governs the unit length of the parametric space with respect to that of the embedded Euclidean space. Such a formulation assumes the parameterization to be piecewise linear on triangles, which is sufficient for our need. Note that ∇u_T, ∇v_T, $\hat{u}_T$ and $\hat{v}_T$ are all 2D vectors expressed in the local frame of triangle T. The latter two are required to be at the center of triangle T, which is in conflict with the fact that the cross field computed is defined only on mesh vertices. This discrepancy is solved by barycentric interpolation to obtain a representation vector at the center of the triangle based on the representation vectors defined on its composing vertices. Having the representation

vector allows the recovery of the corresponding 4-RoSy field value at triangle centers as illustrated in Figure 4.3.3b, which provides the two required orthogonal directions. The optimization also assumes that all singularities in the 4-RoSy field are located on vertices, which is not guaranteed by the 4-RoSy method. Therefore, an additional step is performed to split each triangle containing a singularity into three smaller ones by connecting the original vertices to a newly added vertex at the singularity position. It is impossible for any triangle to contain more than one singularity, as the cross field is assumed piecewise linear, thus guaranteeing that at most one position inside any triangle corresponds to a singular value. The optimization objective is then the global energy defined as the integral of E_T over the entire mesh $\mathcal{M}$

$$E_{\mathcal{M}} = \int_{\mathcal{M}} E_T dA = \sum_{T \in \mathcal{M}} A_T E_T \tag{4.4.2}$$

where A_T is the area of triangle T.

4.4.1 Cut Graph

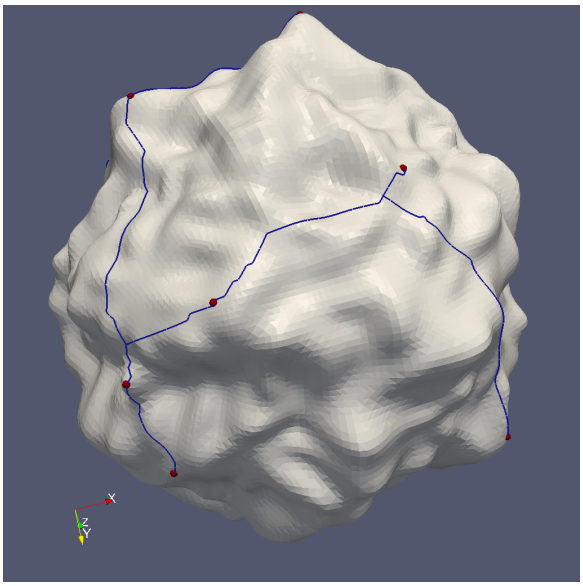
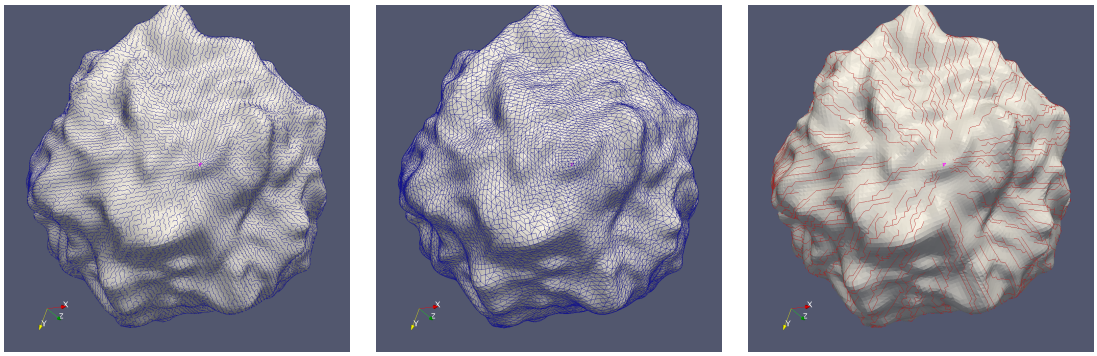


Figure 4.4.1: A Cut Graph of a Gabor Surface

Depending on the topology of the mesh being considered, there may be no solution to the problem of creating an injective map from the mesh to 2D parametric space. For example, there is no way to flatten a sphere onto a plane, while preserving the one-to-one property. In order for

there to be a solution, the mesh has to be transformed into a disk topology. This can be achieved by cutting the mesh open along selected edges, called a cut graph. Such a set of cuts may not disconnect the mesh. The cut graph must include all singularities in the 4-RoSy field, because the cut graph will become the boundary of the mesh in parametric space and these singularities must lie on boundary. This is necessary, since singularities correspond with non valence 4 vertices in the quadrangulation, which would create angular defects if placed at any internal vertex. Figure 4.4.1 shows an example of a valid cut graph for a gabor surface. Note that the graph passes through all singularities (red dots) on the surface, while leaving the surface connected in such a way that it is homeomorphic to a disc. The following steps are used to form a valid cut graph:



(a) Spanning Tree (b) Spanning Tree Edges (c) Full Cut Graph

Figure 4.4.2: Cut Graph Generation

1. Start at a random triangle T, illustrated in Figure 4.4.2 as the central purple triangle.
2. Generate a spanning tree rooted at T containing all triangles as illustrated in Figure 4.4.2a.
3. An edge shared by a pair of triangles crossed by the spanning tree is considered a spanning tree edge as shown in Figure 4.4.2b. The remaining triangle edges form a cut graph for the mesh, as illustrated in red in Figure 4.4.2c.
4. Eliminate all open paths in the cut graph. An open path is a sequence of connected edges where at least one ending vertex is touched only once by the cut graph.
5. Add all paths that connect singularities onto the cut graph, ensuring that all singularities lie on the parametric boundary.

Generating the spanning tree can be done by a breadth first traversal of the mesh triangles starting from the initial triangle T, with connectivity between triangles defined as sharing a common

edge in the mesh. Elimination of open paths needs to be applied iteratively, a process illustrated in

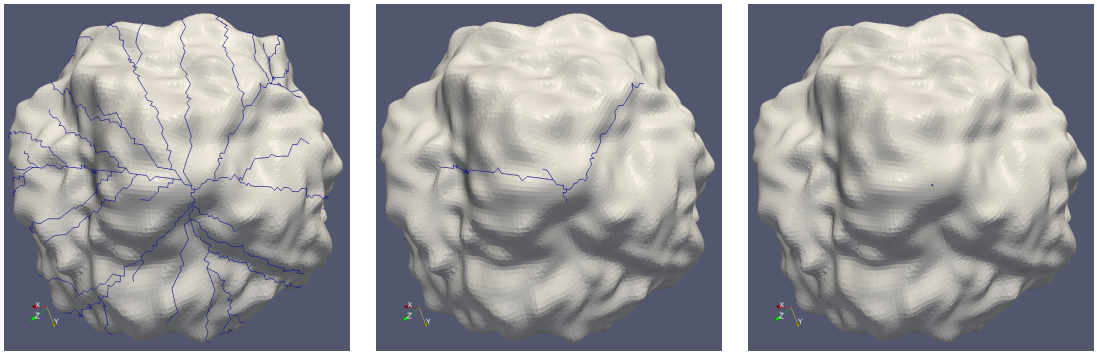


Figure 4.4.3: Open Paths Elimination

Figure 4.4.3. This is so because eliminating an open path might make a closed path open, illustrated as the black and white difference in Figure 4.4.4 where the white edges are those that will remain after the current open paths, shown in black, are eliminated. Note, that this will result in many of the white edges becoming open. The elimination process may shrink the cut graph into a single

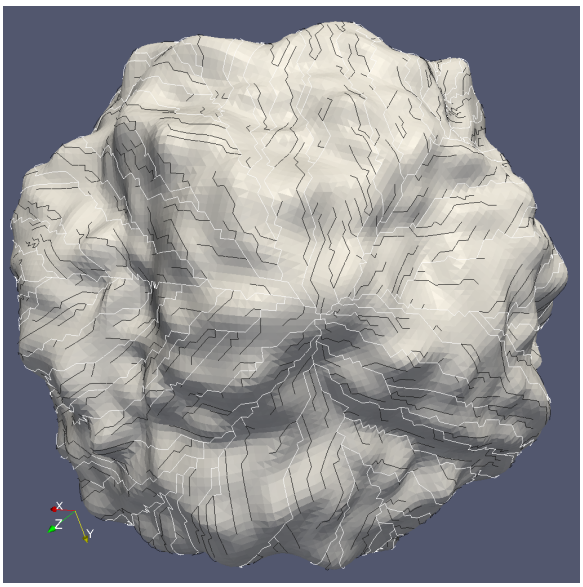


Figure 4.4.4: Eliminating Open Paths Turns Closed Paths Open

point for meshes of sphere topology such as the gabor surfaces used in our previous study, a situation illustrated in Figure 4.4.3 right most image. A single point cut graph is a valid cut graph.

Adding singularities into the cut graph is achieved by applying Dijkstra's shortest path algorithm for every singularity. In the algorithm, the search finishes as soon as a path is found to

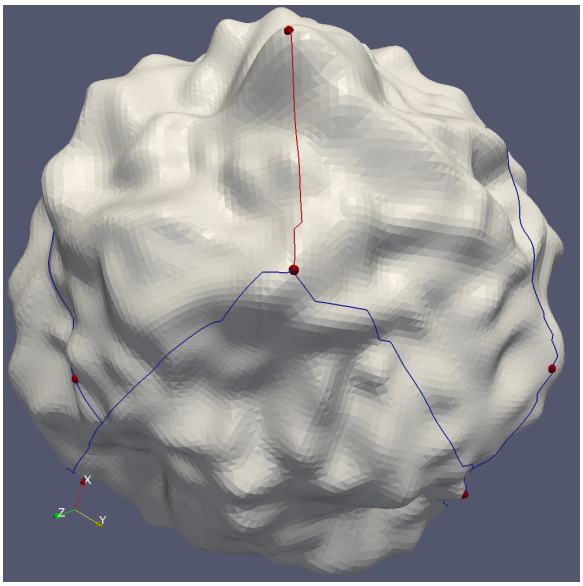


Figure 4.4.5: Connect a Singularity by Expanding the Cut Graph

connect the singularity to the cut graph. The first path to do so is considered shortest and added onto the cut graph. As such, each singularity may see a different cut graph as it is progressively expanded to include previously visited singularities. Figure 4.4.5 illustrates one step in this process where the blue graph is the current cut graph and the red curve is added to the cut graph to connect the top singularity. The cut graph after this iteration is the union of the blue and red curves. Therefore, the order in which singularities are visited affects the shape of the cut graph constructed. All such cut graph shapes are equivalent, for our purpose, as the goal is to cut the mesh into a disk topology, not necessarily of any particular shape.

It is important for the singularity connection process to not introduce additional loops to the cut graph, which, if they occurred, would change the topology of the cut graph, hence the topology of the mesh, invaliding the process. Fortunately, the connection process cannot possibly create loops. This is so because Dijkstra's algorithm can never create a loop, since once a vertex is visited, it will not be revisited. Therefore, the only case where connecting a singularity onto the cut graph could create a loop is when the path connecting the singularity crosses the cut graph more than once, thus potentially closing an open path. Obviously, such a multi-cross path cannot exist because the shortest path search would have terminated on the first cross.

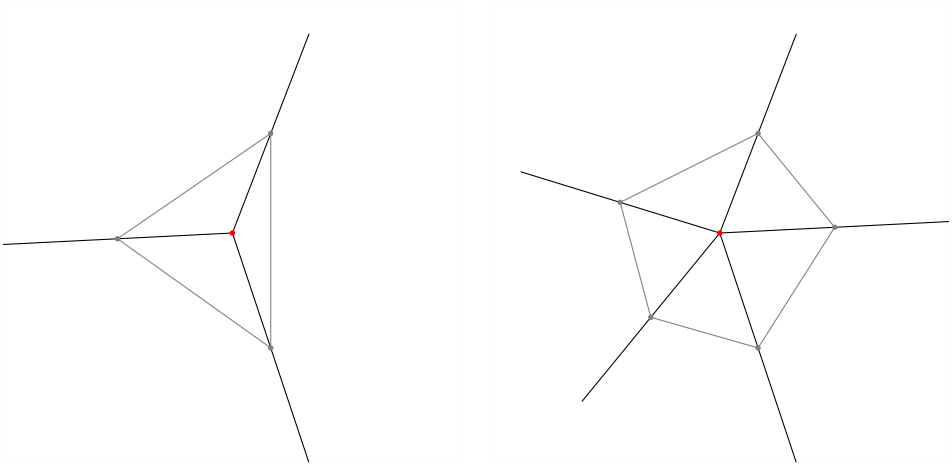


Figure 4.4.6: Singularity on Non-Integer Locations in the Parametric Space

4.4.2 Constraints

There are two types of constraints in the optimization, i.e., integer constraints and linear constraints. Mesh vertices that possess a zero representation vector, i.e., singularities, must be mapped to integer values in the parametric space so that a pure quadrangulation may be obtained through the optimization process. Allowing singularities to be on non-integer locations results in n-sided ($n \neq 4$) faces in quadrangulation which impairs our objective to have a grid-like line texture covering. Figure 4.4.6 illustrates what happens to singularities when they are mapped to non-integer locations in the parametric space. The red dots are the singularities, black lines are separatrices and gray polygons are the non-quadrilateral geometric primitives that will be formed to include the singularities. This is so because an edge can only encode one flow direction based on the piecewise linear assumption. When a singularity is in the interior of a quadrilateral, one of the edges must touch locations on the mesh with different flow behavior, making it impossible for that edge to assume a consistent flow direction. Based on the type of the singularity, the only solution is to either split the edge or merge two connected edges into one, resulting in a pentagon or a triangle. Either one destroys the pure quadrangulation goal. The solution is to have singularities on vertices, i.e., integer locations in the parametric space, thus allowing multiple quadrilaterals to meet at the singularity to capture the different flow behavior. Figure 4.4.7 illustrates such a scenario.

A second type of integer constraint comes from the compatibility requirement across the cut graph boundary. This requirement ensures that the parameterization assigns values to vertices on the two sides of the cut graph in a consistent manner such that a later sampling step generates lines

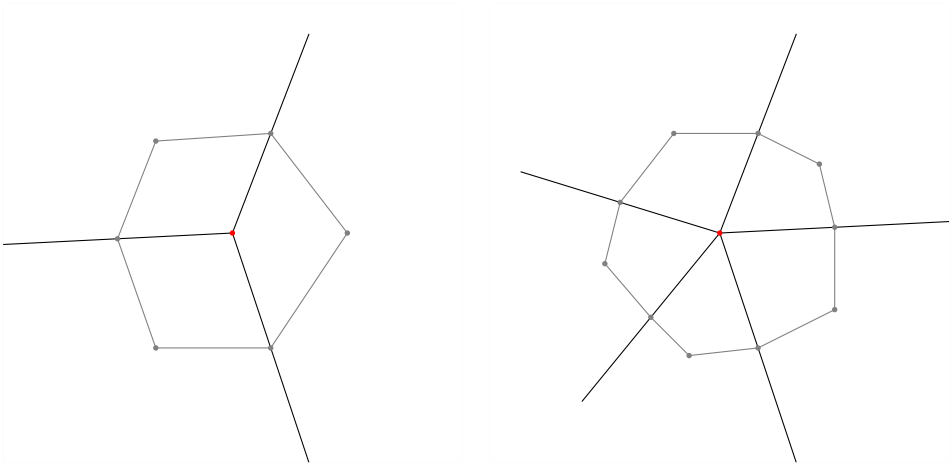


Figure 4.4.7: Singularity on Integer Locations in the Parametric Space

meeting perfectly across the cut graph from both sides. Without this requirement, it is likely the lines generated from one side of the cut graph are offset from lines of the other side, causing visible seams for the line pattern as illustrated in Figure 4.4.8, where the green line is the cut graph, and the red and blue lines are contours in the u, v directions respectively.

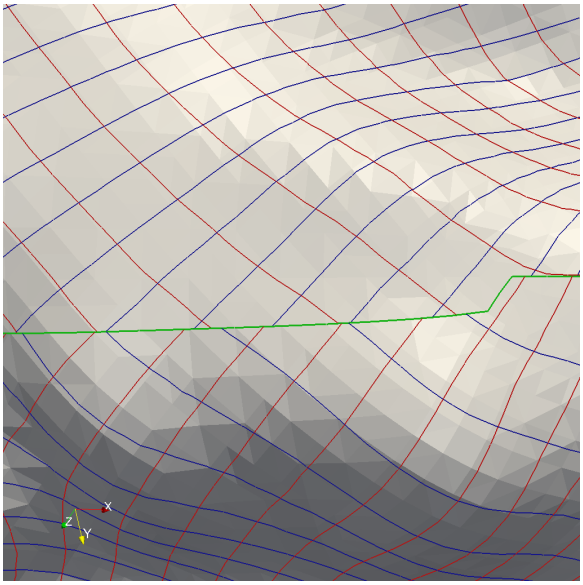


Figure 4.4.8: Visible Seams Across Cut Graph

The compatibility is guaranteed by allowing only a grid automorphism as the transition function across the cut graph. Vertices on the cut graph will necessarily be paired, with identical vertices associated with quadrilaterals on each side of the cut. If these paired vertices have

parameters (u, v) and (u', v'), the parameters must be related by the equality

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = R^i_{90^\circ} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} j \\ k \end{bmatrix} \tag{4.4.3}$$

where i, j, k are all integers and $R^i_{90^\circ}$ represents an integer multiple of 90° degree 2D rotations in the parametric space. Equation 4.4.3 provides integer constraints as well as linear constraints, specifically equality constraints, which are the only type of linear constraints in the optimization. Since the cut graph has a discrete representation as a sequence of connected mesh edges, the transition function is formed on a per-cut-edge basis, which provides a concise formation of the equality constraints. In particular, for each cut edge e, a set of integer parameters (j_e, k_e) is defined and the compatibility requirement of the incident vertices p and q is formed as

$$\begin{aligned} \begin{bmatrix} u'_p \\ v'_p \end{bmatrix} &= R^{i_e}_{90^\circ} \begin{bmatrix} u_p \\ v_p \end{bmatrix} + \begin{bmatrix} j_e \\ k_e \end{bmatrix} \\ \begin{bmatrix} u'_q \\ v'_q \end{bmatrix} &= R^{i_e}_{90^\circ} \begin{bmatrix} u_q \\ v_q \end{bmatrix} + \begin{bmatrix} j_e \\ k_e \end{bmatrix} \end{aligned}. \tag{4.4.4}$$

i_e can be fixed by propagating a consistent orientation for all triangles on the mesh, a process done by randomly starting from an interior triangle, one that does not touch the cut graph, and propagating its cross orientation to its neighboring triangles in a breadth first traversal that eventually includes all mesh triangles. The traversal guarantees a complete coverage, since the mesh is connected even after the cut. The propagation establishes a zero-rotation for all the inner edges, leaving any inconsistency on the cut edges only. After that, i_e for each cut edge e can be found by comparing the cross orientation on both sides of the edge.

4.4.2.1 Consistency Propagation

The following steps are used to achieve the consistency propagation:

1. Mark all faces as unaligned.

2. Select a random inner face to start. An inner face is a mesh face that does not contain any cut edges or cut vertices.

3. In a breadth first order, enumerate all neighboring faces of the current face f. For each face f' enumerated,

   (a) If it has been aligned, then do nothing.

   (b) Otherwise, identify the edge e it shares with f and

      i. If e is a cut edge, then do nothing.

      ii. Otherwise, align the cross of f' with that of f and mark f' as aligned, a process detailed below.

The cross of a face is a set of 4 unit vectors with no intrinsic order between these vectors. Moreover, because the cross is generated as a 4-way rotational symmetry, having any one of member vectors allows the complete reconstruction of the entire cross. For the optimization, only two of the four the member vectors are needed to define the local guiding frame. This causes a problem when the two vectors picked from adjacent triangles, and the frame formed, are inconsistent. The inconsistency makes the optimization problem harder to solve, thus leading to less optimal solutions which degrade, or even destroy, the quality of the lines constructed later. Therefore, adjacent triangles must align their frames in a consistent manner. Ideally, having the cross in the target triangle to be identical to that of the source triangle would ensure perfect alignment. However, this is impractical for two reasons:

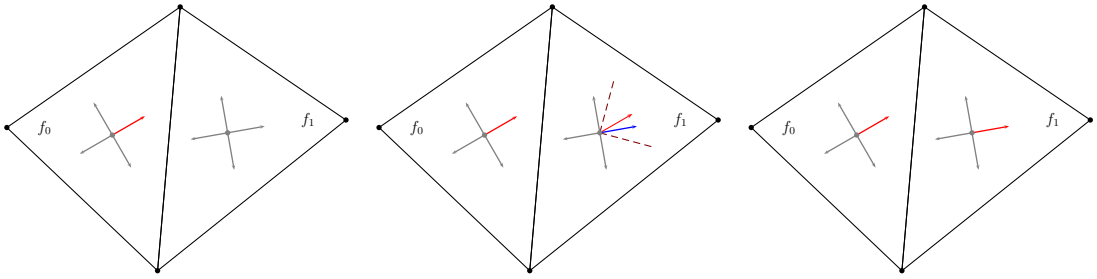
1. The exact duplicate of the source cross in the target triangle may be different from the target cross that is already in place. Requiring perfect alignment demands replacing the target cross with the source cross duplicate, thus altering the existing target cross. This is unacceptable as the consistency propagation should not change the global cross field of the mesh.

2. Perfect alignment strives to optimize local consistency, at the potential cost of global consistency. The consequence is that triangles visited at a later phase of the breadth first traversal might receive conflicting alignment requirements from its adjacent neighbors that cannot be resolved, because the global consistency has been broken by a sequence of local optimizations.

The reason consistency propagation stops at the cut graph (i.e., if two faces share a cut edge they do not attempt to align their crosses), is because, for general cases, there will always be incompatibility in the cross field and such incompatibility has to exist somewhere on the mesh.

Restricting them to the cut graph makes it possible to capture them by transition functions, thus allowing the optimization to eliminate their effect of causing visible seams in the parameterization.

4.4.2.2 Consistent Orientation



(a) Before Alignment (b) Propagating Orientation (c) After Alignment

Figure 4.4.9: Propagating Consistent Orientation

To align the cross of face f_1 with that of f_0 is to rearrange the order of the member vectors of face f_1, such that in an ordered traversal, the corresponding vectors from both crosses form a minimum angle as illustrated in Figure 4.4.9 and listed below:

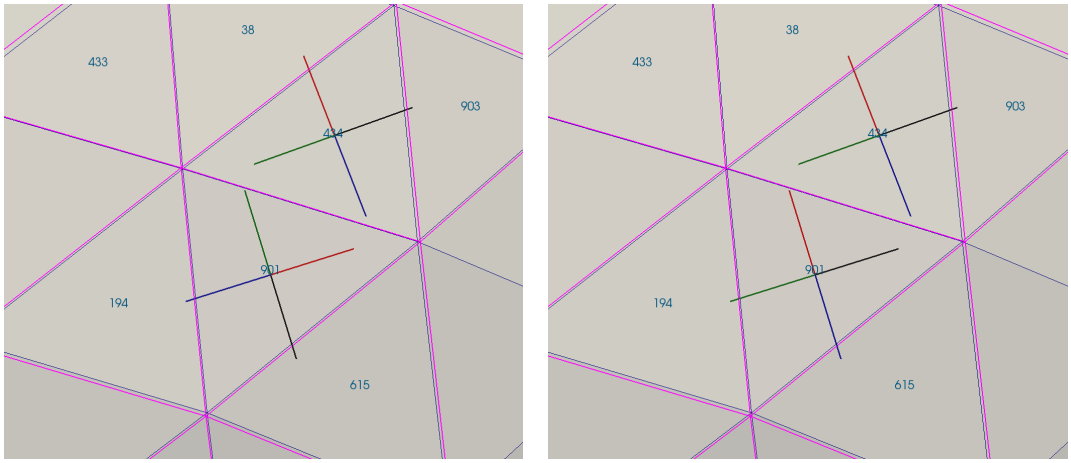
1. Map the 1^{st} member vector (red arrow in f_0 in Figure 4.4.9a) of the cross in f_0 into the local frame of f_1, denoted as $\hat{r}_{f0\mapsto f1}$ (red arrow in f_1 in Figure 4.4.9b).

2. Identify $i \in \{1, 2, 3, 4\}$ such that the i^{th} member vector (blue arrow in Figure 4.4.9b) of f_1 is inside the 45° cone spanned by $\hat{r}_{f0\mapsto f1}$ (dark red cone in Figure 4.4.9b). Such an i is guaranteed to exist due to the rotational symmetric property of the cross. This i^{th} member vector of f_1 is considered the best match to the 1^{st} member vector of f_0.

3. Stably reorder f_1's member vectors such that the original i^{th} member vector becomes the 1^{st} member vector in the new ordering as illustrated in Figure 4.4.9c.

This approach guarantees that, the target cross is not altered by the alignment, thus maximizing local alignment without affecting global consistency. Figure 4.4.10 shows the same cross before and after consistency propagation where red bars represent the 1^{st} member vectors in the two faces respectively.

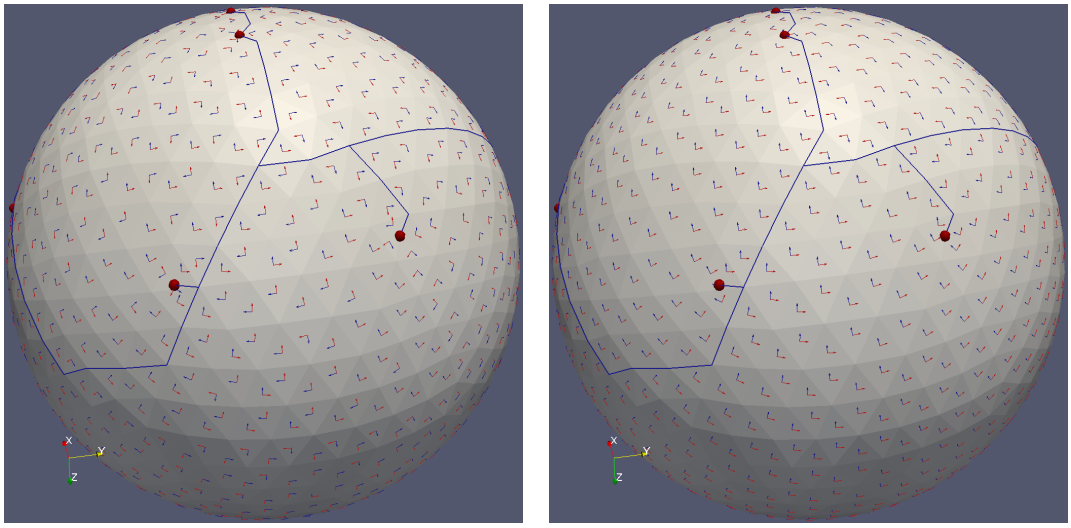
After the orientation consistency has been propagated, all inner edges have a zero rotation and the integer i_e on a cut edge e can be computed by comparing the two crosses of e's neighboring



(a) Before (b) After

Figure 4.4.10: Consistency Propagation from f_{434} to f_{901}

triangles in a way essentially the same as the consistency propagation. The only difference is that when the best match member vector is found in the target triangle, its index gives the number of 90° rotations between the two crosses, i.e., i_e of the transition function from the source triangle to the target one. Figure 4.4.11 shows the cross field before and after consistency propagation for a spherical triangle mesh. The red spheres are singularities and the blue curve constitutes the cut graph. Red arrows give the direction of the first member vector and blue the second. The consistency in the right image is easily visible in comparison with the left one.



(a) Before (b) After

Figure 4.4.11: Consistency Propagation for the Entire Mesh

4.4.3 Objective Setup

For the quadratic objective defined in Equation 4.4.2 to be minimized by the mixed-integer solver [7], it has to be expressed in the standard form

$$\begin{aligned} \min \quad & E = \tfrac{1}{2}\vec{x}'\mathbf{Q}\vec{x} - \vec{b}'\vec{x} \\ \text{s.t.} \quad & \mathbf{C}\vec{x} = \mathbf{0} \\ & x_i \in \mathbb{Z} \quad \forall\, i \in \mathbb{I} \end{aligned} \tag{4.4.5}$$

where E is the quadratic energy to be minimized, $\vec{x}$ is the n-dimensional solution vector which receives the minimizer and $(\cdot)'$ forms the transpose for its vector or matrix operand. $\mathbf{Q}$ is an $n \times n$ symmetric matrix that determines the quadratic term and $\vec{b}$ is an n-dimensional vector that governs the linear term. $\mathbf{C}$ is an $m \times (n+1)$ matrix that encodes the equality constraints, where m is the number of linear constraints and the additional column dimension in $\mathbf{C}$ is to capture non-zero equality constraints. The set $\mathbb{I}$ is an index set determining the index of all variables x_i whose solutions are constrained to be integer, where x_i is the i^{th} variable of the solution vector. As such, $\mathbb{I}$ determines the integer constraints and makes the optimization a mixed-integer problem.

The quadratic objective as defined in Equation 4.4.5 is an overdetermined problem in that $\mathbf{Q}$ is a singular matrix and there are redundant variables in $\vec{x}$. This is so because the system is subject only to equality constraints captured by $\mathbf{C}$, which the solver will use to reduce the dimensionality of $\mathbf{Q}$ and $\vec{b}$ before trying to optimize. The reduced matrix $\tilde{\mathbf{Q}}$ is symmetric positive definite, a property governed by the class of the optimization problem being solved. An intuitive understanding of the mixed-integer solver is that it first projects the overdetermined problem onto a lower dimensional space by utilizing the hyper-plane conditions presented in $\mathbf{C}$. After the projection, the optimization becomes an unconstrained one by ignoring the integer constraints, which can be efficiently solved by solving the linear system that is the stationary point of the quadratic system. As this lower dimensional space is spanned by the maximal linearly independent group of $\mathbf{Q}$'s composing vectors, the coefficient matrix of this linear system is non-singular symmetric positive definite with several fast solvers available. Based on the initial solution, integer constraints are added back one at a time, with every addition requiring the solution of two sub linear systems for the two closest integers around the chosen continuous variable. Having this setup, it is important to encode Equation 4.4.2 into a matrix, a process shown in the following analysis.

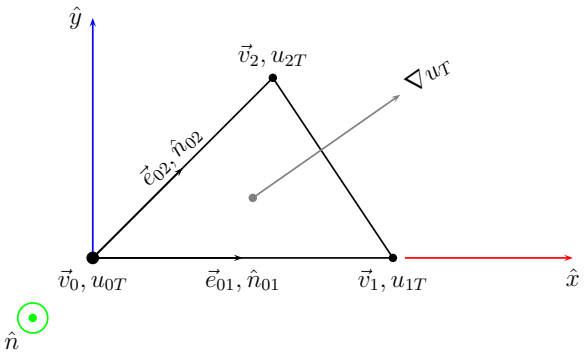


Figure 4.4.12: Gradient of u Parameter

For a triangle T as illustrated in Figure 4.4.12, a local 3D coordinate frame $\{\hat{x}, \hat{y}, \hat{n}\}$ can be constructed by picking one of its vertices as the origin ($\vec{v}_0$) and forming axes along one of its edges ($\vec{e}_{01}$) and the other orthogonal direction ($\hat{y}$). The cross field is considered piecewise linear as well as the parameterization sought, which means the gradient vector ∇u_T of parameter u in T is constant at any point of triangle T. As such, the directional derivatives of u on the two edge directions $\vec{e}_{01}$ and $\vec{e}_{02}$ are given by

$$\begin{aligned} \langle \hat{n}_{01}, \nabla u_T \rangle = \frac{u_{1T} - u_{0T}}{\|\vec{e}_{01}\|} \quad &\Rightarrow \quad \langle \vec{e}_{01}, \nabla u_T \rangle = u_{1T} - u_{0T} \\ \langle \hat{n}_{02}, \nabla u_T \rangle = \frac{u_{2T} - u_{0T}}{\|\vec{e}_{02}\|} \quad &\Rightarrow \quad \langle \vec{e}_{02}, \nabla u_T \rangle = u_{2T} - u_{0T} \end{aligned} \tag{4.4.6}$$

where $\hat{n}_{0i}$ is the unit vector parallel to $\vec{e}_{0i}$ for $i = 1, 2$ respectively and u_{iT} is the u parameter value at the $i + 1^{\text{st}}$ corner of triangle T for $i = 0, 1, 2$. The above relation is captured in matrix form as

$$\begin{bmatrix} \vec{e}_{01}{}' \\ \vec{e}_{02}{}' \end{bmatrix} \nabla u_T = \begin{bmatrix} u_{1T} - u_{0T} \\ u_{2T} - u_{0T} \end{bmatrix} \quad \Rightarrow \quad \nabla u_T = \begin{bmatrix} \vec{e}_{01}{}' \\ \vec{e}_{02}{}' \end{bmatrix}^{-1} \begin{bmatrix} u_{1T} - u_{0T} \\ u_{2T} - u_{0T} \end{bmatrix} \tag{4.4.7}$$

which gives an explicit expression for ∇u_T with respect to the u parameter values at the three corners and the edge vectors in T's local frame. The inverse matrix in the above equation, denoted as $\mathbf{M}_T$ hereafter, must exist because $\vec{e}_{01}$ and $\vec{e}_{02}$ are linearly independent in the local frame unless the triangle is in a degenerate state, which is not admitted by the optimization. Equation 4.4.7 can

be further transformed into

$$\nabla u_T = \mathbf{M}_T \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{0T} \\ u_{1T} \\ u_{2T} \end{bmatrix} = \mathbf{M}_T \mathbf{T}_T \tilde{u}_T \tag{4.4.8}$$

which allows the u parameters and $\mathbf{M}$ and $\mathbf{T}$ from different triangles to be stacked to form a complete vector and matrix representation for the entire mesh. The same applies to the v parameters, leading to the following equation

$$\nabla v_T = \mathbf{M}_T \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{0T} \\ v_{1T} \\ v_{2T} \end{bmatrix} = \mathbf{M}_T \mathbf{T}_T \tilde{v}_T. \tag{4.4.9}$$

Since ∇u_T and ∇v_T share the same $\mathbf{M}_T$ and $\mathbf{T}_T$, they may be encoded more concisely as

$$\begin{bmatrix} \nabla u_T \\ \nabla v_T \end{bmatrix} = \begin{bmatrix} \mathbf{M}_T & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_T \end{bmatrix} \begin{bmatrix} \mathbf{T}_T & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_T \end{bmatrix} \begin{bmatrix} \tilde{u}_T \\ \tilde{v}_T \end{bmatrix} = \tilde{\mathbf{M}}_T \tilde{\mathbf{T}}_T \tilde{x}_T \tag{4.4.10}$$

where $\tilde{x}'_T = (\tilde{u}_T, \tilde{v}_T)'$ is a representation of the (u, v) parameters at all corners of triangle T. With that, Equation 4.4.1 is transformed into matrix form by the following process:

$$\begin{aligned} E_T &= \|h\nabla u_T - \hat{u}_T\|^2 + \|h\nabla v_T - \hat{v}_T\|^2 \\ &= \langle h\nabla u_T, h\nabla u_T\rangle - 2\langle h\nabla u_T, \hat{u}_T\rangle + \langle \hat{u}_T, \hat{u}_T\rangle \\ &+ \langle h\nabla v_T, h\nabla v_T\rangle - 2\langle h\nabla v_T, \hat{v}_T\rangle + \langle \hat{v}_T, \hat{v}_T\rangle \\ &= h^2(\tilde{\mathbf{M}}_T\tilde{\mathbf{T}}_T\tilde{x}_T)'\tilde{\mathbf{M}}_T\tilde{\mathbf{T}}_T\tilde{x}_T - 2h(\tilde{\mathbf{M}}_T\tilde{\mathbf{T}}_T\tilde{x}_T)'\hat{x}_T + \langle \hat{x}_T, \hat{x}_T\rangle \\ &= h^2\tilde{x}'_T\tilde{\mathbf{T}}'_T\tilde{\mathbf{M}}'_T\tilde{\mathbf{M}}_T\tilde{\mathbf{T}}_T\tilde{x}_T - 2h\tilde{x}'_T\tilde{\mathbf{T}}'_T\tilde{\mathbf{M}}'_T\hat{x}_T + \langle \hat{x}_T, \hat{x}_T\rangle \end{aligned} \tag{4.4.11}$$

where $\hat{x}'_T = (\hat{u}_T, \hat{v}_T)'$ is a representation of the guiding frame at the center of triangle T. Note $\hat{x}_T$ is not a unit vector, the $\hat{}$ notation is used here merely to indicate its relation to the two unit vectors $\hat{u}_T$ and $\hat{v}_T$ that form the frame. Having Equation 4.4.11 allows the expression of Equation 4.4.2 in matrix form as

$$\begin{aligned} E_{\mathcal{M}} &= \sum_{T\in\mathcal{M}} A_T E_T \\ &= h^2\vec{x}'\mathbf{T}'\mathbf{M}'\mathbf{A}\mathbf{M}\mathbf{T}\vec{x} - 2h\vec{x}'\mathbf{T}'\mathbf{M}'\mathbf{A}\hat{x} + \langle \hat{x}, \hat{x}\rangle \end{aligned} \tag{4.4.12}$$

where $\vec{x}$ is $\tilde{x}_T$ stacked for all triangles in the order of ascending indices, $\mathbf{A}$ is a diagonal matrix formed by all triangle areas, $\mathbf{M}$ is a block diagonal matrix with $\tilde{\mathbf{M}}_T$ placed in order along the diagonal as building blocks and $\mathbf{T}$ has a similar layout as $\mathbf{M}$, $\hat{x}$ gathers all $\hat{u}_T$ and $\hat{v}_T$ and stacks them in order (it is not a unit vector). Assuming $\mathcal{M}$ contains N_T triangles, the dimension of $\mathbf{A}$ is $4N_T \times 4N_T$, $\mathbf{M}$ is $4N_T \times 4N_T$, $\mathbf{T}$ is $4N_T \times 6N_T$, $\vec{x}$ is $6N_T \times 1$ and $\hat{x}$ is $4N_T \times 1$.

Since $\langle \hat{x}, \hat{x} \rangle$ is a constant term, the minimizer of $E_{\mathcal{M}}$ is the same with or without it, a fact that reduces $E_{\mathcal{M}}$ to the standard form as given in Equation 4.4.5 with $\mathbf{Q} = h^2\mathbf{T}'\mathbf{M}'\mathbf{A}\mathbf{M}\mathbf{T}$ and $\vec{b} = h\mathbf{T}'\mathbf{M}'\mathbf{A}\hat{x}$. $\mathbf{A}$ is an invertible symmetric matrix, $\mathbf{M}$ is invertible but generally asymmetric, and $\mathbf{T}$ is non-square and asymmetric. $\mathbf{Q}$, $\vec{b}$ and $\vec{x}$ require further adjustment, however, because they have to share the same solution vector $\vec{x}$ with the constraint matrix $\mathbf{C}$. Since $\mathbf{C}$ is the equality constraint matrix, it has to capture the equality relations for all cut edges as described in Equation 4.4.4, requiring that $\vec{x}$ be expanded to include $2N_e$ additional integer variables, where N_e is the number of non-boundary edges in the cut graph. This is easily done by stacking (j_e, k_e) variables for all such edges in $\vec{x}$ after the (u, v) parameters for all triangle corners in ascending index order for some ordering of non-boundary cut edges. Expansion of the solution vector changes its dimension to $6N_t + 2N_e$, which requires corresponding adjustment to $\mathbf{Q}$ and $\vec{b}$ so that the multiplication with $\vec{x}$ remains valid. One way to perform the adjustment is to express $\mathbf{Q}$ and $\vec{b}$ as

$$\begin{aligned} \mathbf{Q} &= \mathbf{B}'\mathbf{A}\mathbf{B} \\ \vec{b} &= \mathbf{B}'\mathbf{A}\hat{x} \end{aligned} \tag{4.4.13}$$

where $\mathbf{B} = h\mathbf{M}\mathbf{T}$ and is expanded with $\mathbf{0}$ on its right such that $\mathbf{B}$ is $4N_T \times (6N_T + 2N_e)$ in dimension. Such an expansion clearly makes $\mathbf{Q}$ non-invertible, a problem reconciled by the fact that the mixed-integer solver uses the equality constraints to reduce $\mathbf{Q}$ before any attempts to solve the system. It is possible to use the equality constraints to form an invertible quadratic matrix with a solution vector containing all the (j_e, k_e) before feeding it to the solver, but that merely moves the variable elimination step from within the solver to be out of it, with no clear benefits to the overall optimization process.

4.4.4 Constraint Setup

As described in Section 4.4.2, the optimization is subject to integer and equality constraints. In particular, all singularities must be mapped onto integer locations in the parametric space and all (j_e, k_e) pairs on non-boundary cut edges are required to be integers as well. This means that, for every triangle of index t whose $c + 1^{\text{st}}$ corner corresponds to a singularity vertex of the mesh ($c \in [0, 2]$), the set $\mathbb{I}$ in 4.4.5 must contain $6t + c$ and $6t + c + 3$, which are the indices of the (u, v) parameters for the corner in the solution vector $\vec{x}$ in Equation 4.4.5, and where $t \in [0, N_T)$ with N_T being the total number of triangles in the mesh. For every non-boundary cut edge of index e in a sequential ordering of all non-boundary cut edges, set $\mathbb{I}$ also needs to include $6N_T + 2e$ and $6N_T + 2e + 1$ as they are the indices of (j_e, k_e) for the cut edge into the solution vector $\vec{x}$. All indices are 0 based, which means the first element of any sequential ordering receives an index 0.

Apart from the integer constraints, for every edge in the mesh that is shared by two triangles of indices t_0 and t_1, each of its composing vertices contributes two equality constraints involving the (u, v) parameters of its associated corners in the two triangles. Specifically, for each vertex $\vec{v}$ of the edge,

1. if the edge is not a cut edge, then $\mathbf{C}$ must contain a row with

$$\begin{aligned} [6t_0 + c_0] &= 1 \\ [6t_1 + c_1] &= -1 \end{aligned}$$

   and another row with

$$\begin{aligned} [6t_0 + c_0 + 3] &= 1 \\ [6t_1 + c_1 + 3] &= -1 \end{aligned}$$

   where c_0 and c_1 are the indices of the corners in t_0 and t_1 that correspond to $\vec{v}$ respectively, and the $[i]$ notation denotes the $i + 1^{\text{st}}$ element of the row;

2. otherwise, the relation between the (u, v) parameters of c_0 and c_1 must involve the transition function on the edge as defined in Equation 4.4.4. Letting e be the index of the edge in the

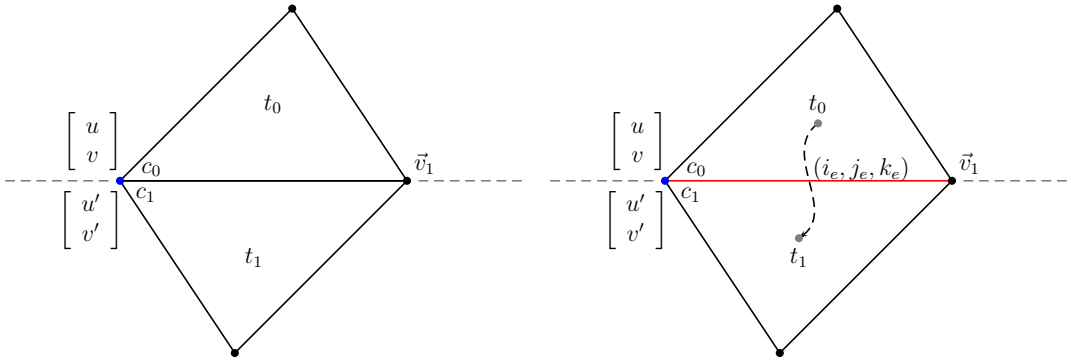
cut graph non-boundary edge sequence, $\mathbf{C}$ must contain a row with

$$\begin{aligned}
[6t_0 + c_0] &= \cos(i_e 90°) \\
[6t_0 + c_0 + 3] &= -\sin(i_e 90°) \\
[6N_t + 2e] &= 1 \\
[6t_1 + c_1] &= -1
\end{aligned}$$

and another row with

$$\begin{aligned}
[6t_0 + c_0] &= \sin(i_e 90°) \\
[6t_0 + c_0 + 3] &= \cos(i_e 90°) \\
[6N_t + 2e + 1] &= 1 \\
[6t_1 + c_1 + 3] &= -1
\end{aligned}$$

This formulation is a direct expansion of Equation 4.4.4 based on the structure of the solution vector $\vec{x}$ as defined in Equation 4.4.12. Since i_e must always be an integer, one of $\cos(i_e 90°)$ and $\sin(i_e 90°)$ must be 0, in which case, there is no need to specify that element as all unspecified elements of $\mathbf{C}$ receive a zero value automatically. Figure 4.4.13 gives a visual presentation showing constraints on



(a) On an Inner Edge (b) On a Cut Edge (Transition from t_0 to t_1)

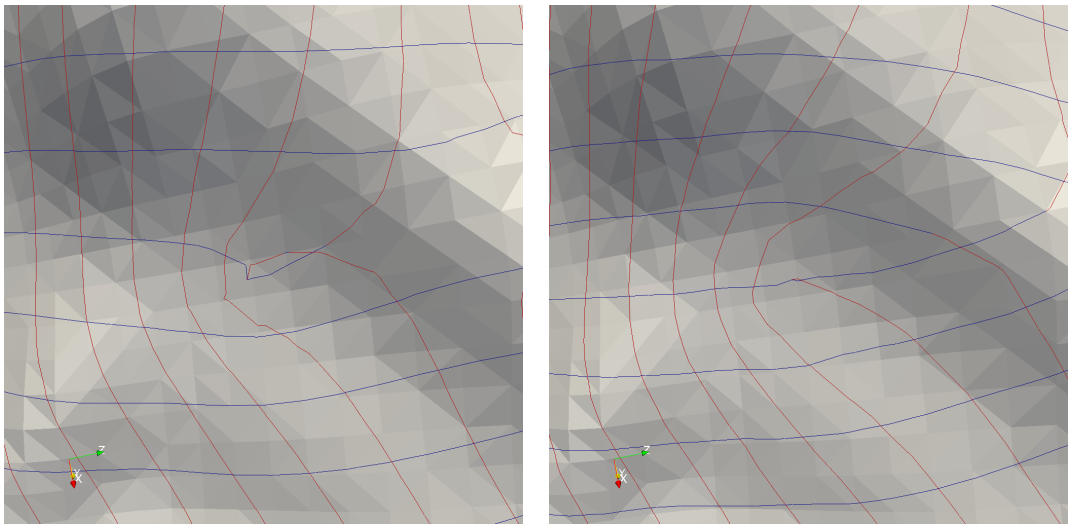
Figure 4.4.13: Constraints on Edges

inner edges and on cut edges. The blue dot is the vertex currently being examined, c_0 and c_1 are the corners that correspond to the vertex in the two adjacent triangles sharing it and the red edge is a cut edge. (u, v) and (u', v') are the parameters for c_0 and c_1. For a regular mesh edge, (u, v) and (u', v') must have identical values and for a cut edge, they must satisfy the transition function of the edge as illustrated by the right figure.

4.4.5 Choosing h

The h parameter in Equation 4.4.5 controls the size of the quadrilateral. It is important to choose a proper value for h, because if h is too small, the resulting density of the grid-like line structure will be too large to be useful. On the other hand, if h is too large, there might not even be a solution to the optimization problem, because the singularities are constrained on integer locations, and when h is too large to fit a quadrilateral between two singularities, then it is impossible to have both of them on integer locations in the parametric space. This will result in a poor optimization solution, upon which no useful line structure can be generated. According to our experience, a proper value for h is some integer fraction, e.g. $1/2$ or $1/4$, of the smallest distance between the singularities of the guiding frame field. Ideally, the geodesic distance should be used to measure singularity separation. In case that is too expensive to compute, Euclidean distance has also been found to provide reasonably good line quality.

4.4.6 Local Stiffening



(a) Without (b) Applied

Figure 4.4.14: Local Stiffening

The parameterization is the result of a global minimization of the orientation energy integrated over the entire mesh. It is possible for some triangles to be trapped in a local state with high metric distortion in their vicinity as illustrated in Figure 4.4.14a, which can be visually unpleasant and confusing. In this case, local stiffening can be employed to penalize such conditions in the

optimization process so that the system rests on a global optimum, with less distortion, at the cost of a higher objective energy, as illustrated in Figure 4.4.14b. The local stiffening is incorporated into the optimization by the modified objective energy

$$E_{\mathcal{M}} = \sum_{T \in \mathcal{M}} \omega(T) A_T E_T \tag{4.4.14}$$

where $\omega(T)$ is a weight matrix to penalize high local distortion. This also requires an iterative solution to the optimization problem in which the weight matrix $\omega(T)$ is updated after each optimization and used to guide the next round.

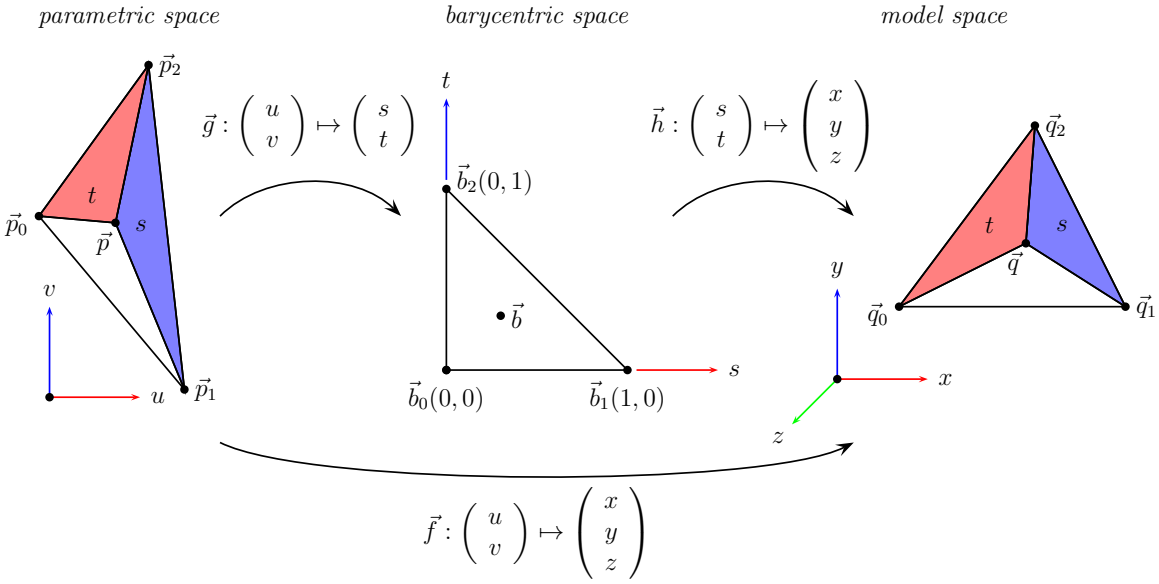


Figure 4.4.15: Mapping from Parametric Space to Model Space

The metric distortion is represented by the singular values of the Jacobi matrix as defined in [30] by Hormann *et al.* Specifically, the Jacobi matrix

$$J_{\vec{f}} = \left(\vec{f}_u, \vec{f}_v \right) \tag{4.4.15}$$

is the partial derivative of $\vec{f}$ with respective to the u and v dimension of the parametric space, where $\vec{f}(u,v)$ maps a point (u,v) from the parametric space to a point (x,y,z) on the mesh in the model space. Based on the piecewise linear assumption, the mapping $\vec{f}$ involves a barycentric interpolation as a middle step as illustrated in Figure 4.4.15. Therefore, $\vec{f}$ is composed of two mappings $\vec{h} \circ \vec{g}$,

$\vec{g} : R^2 \mapsto R^2$ which maps from the parametric space to the barycentric space, and $\vec{h} : R^2 \mapsto R^3$ which maps from the barycentric space to the model space, where

$$\vec{g}\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \frac{(\vec{p}-\vec{p}_2)\times(\vec{p}_1-\vec{p}_2)}{(\vec{p}_0-\vec{p}_2)\times(\vec{p}_1-\vec{p}_2)} \\ \frac{(\vec{p}_0-\vec{p}_2)\times(\vec{p}-\vec{p}_2)}{(\vec{p}_0-\vec{p}_2)\times(\vec{p}_1-\vec{p}_2)} \end{bmatrix} = \frac{1}{(\vec{p}_0-\vec{p}_2)\times(\vec{p}_1-\vec{p}_2)} \begin{bmatrix} (\vec{p}-\vec{p}_2)\times(\vec{p}_1-\vec{p}_2) \\ (\vec{p}_0-\vec{p}_2)\times(\vec{p}-\vec{p}_2) \end{bmatrix} \tag{4.4.16}$$

and

$$\vec{h}(s,t) = s\vec{q}_0 + t\vec{q}_1 + (1-s-t)\vec{q}_2. \tag{4.4.17}$$

The $\times$ operator forms the 2D cross product as

$$\vec{a} \times \vec{b} = a_0 b_1 - a_1 b_0$$

where $\vec{a} = (a_0, a_1)'$ and $\vec{b} = (b_0, b_1)'$ are 2D vectors. By the chain rule, Equation 4.4.15 is transformed into

$$J_{\vec{f}} = J_{\vec{h}} J_{\vec{g}} \tag{4.4.18}$$

where

$$J_{\vec{h}} = \begin{bmatrix} \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} = \begin{bmatrix} x_0 - x_2 & x_1 - x_2 \\ y_0 - y_2 & y_1 - y_2 \\ z_0 - z_2 & z_1 - z_2 \end{bmatrix} \tag{4.4.19}$$

and

$$J_{\vec{g}} = \begin{bmatrix} \frac{\partial s}{\partial u} & \frac{\partial s}{\partial v} \\ \frac{\partial t}{\partial u} & \frac{\partial t}{\partial v} \end{bmatrix} = \frac{1}{A} \begin{bmatrix} v_1 - v_2 & u_2 - u_1 \\ v_2 - v_0 & u_0 - u_2 \end{bmatrix} \tag{4.4.20}$$

and

$$A = (\vec{p}_0 - \vec{p}_2) \times (\vec{p}_1 - \vec{p}_2) = (u_0 - u_2)(v_1 - v_2) - (u_1 - u_2)(v_0 - v_2) \tag{4.4.21}$$

is twice the signed area of the triangle in the parametric space, with a negative sign indicating a flipped orientation. The singular values σ_1, σ_2 of $J_{\vec{f}}$ are the eigenvalues of the first fundamental form

$$\mathbf{I}_{\vec{f}} = J'_{\vec{f}} J_{\vec{f}} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \tag{4.4.22}$$

which gives

$$\sigma_{1,2} = \lambda_{1,2} = \frac{1}{2}\left[(E+G) \pm \sqrt{4F^2 + (E-G)^2}\right]. \tag{4.4.23}$$

The local distortion is then measured as

$$\lambda = |\tau\frac{\sigma_1}{h} - 1| + |\tau\frac{\sigma_2}{h} - 1| \tag{4.4.24}$$

where τ is the sign of A as defined in Equation 4.4.21 and the weight of a triangle T is updated by

$$\omega(T) = \omega(T) + \min\{c|\Delta\lambda(T)|, d\} \tag{4.4.25}$$

where $\Delta\lambda(T)$ is the Laplacian of λ at T, and $c = 1$, $d = 5$ are tunable parameters to constrain the penalization effect. There are several definitions for the discrete Laplace operator, of which the following is used

$$\Delta\delta(\vec{v}_i) = \frac{1}{|N(\vec{v}_i)|}\sum_{\vec{v}_j \in N(\vec{v}_i)}[\delta(\vec{v}_i) - \delta(\vec{v}_j)] \tag{4.4.26}$$

where $\delta(\vec{v})$ is some scalar quantity defined on vertex $\vec{v}$, $\vec{v}_i$ is the vertex whose Laplacian is being computed, $N(\vec{v}_i)$ enumerates all incident vertices of $\vec{v}_i$, with a total number of $|N(\vec{v}_i)|$ elements. The given Laplace operator is defined with respect to mesh vertices, which implies $\Delta\lambda$ is evaluated in the dual mesh, and for a triangle mesh $\mathcal{M}$ with no boundary, $|N(\vec{v})| \equiv 3\ \forall\, \vec{v} \in \mathcal{M}$.

The local stiffening is incorporated as in Equation 4.4.14, the matrix form for the modified optimization problem is

$$\begin{aligned} \mathbf{Q} &= \mathbf{B'WAB} \\ \vec{b} &= \mathbf{B'WA}\hat{x} \end{aligned} \tag{4.4.27}$$

Compared with Equation 4.4.13, the only change is the additional multiplication by $\mathbf{W}$ which is a square diagonal and invertible matrix whose diagonal elements are the weights $\omega(T)$ for all triangles listed in ascending order of triangle indices. For the first iteration, the weight matrix is set to the identity matrix.

4.4.7 Sampling

After the optimization is finished, the solution vector $\vec{x}$ records a (u, v) pair for all corners of all triangles. This defines two scalar functions on the mesh vertices. The process to construct two sets of orthogonal lines samples at integer locations in the parametric space in its two axis directions respectively, thus mapping the integer grid of the parametric space onto the original mesh. The mapping is effectively a contouring algorithm, as it seeks all points $\vec{v}$ on the mesh such that their (u, v) parameters have the integer values being examined, a process very similar in spirit to the Marching Cubes algorithm. Therefore, a Marching Cubes inspired algorithm is used to generate the grid lines.

Given a scalar function $u(\vec{v})$ defined on mesh vertices and an integer isovalue value u_c, for each triangle in the mesh, the u values are compared with u_c at its three corners. Depending on the number of exact matches, the algorithm performs one of the following actions:

0. There is no exact match. For each edge of the triangle, inspect the u values u_0, u_1 on its incident corners. If u_0 and u_1 fall above and below u_c, then there is an intersection of the scalar function with the edge. In this case, compute $t = \frac{u_c - u_0}{u_1 - u_0}$, which is guaranteed to be in $[0, 1]$, then compute the intersection by $\vec{v} = (1 - t)\vec{v}_0 + t\vec{v}_1$ where $\vec{v}_0$ and $\vec{v}_1$ are corresponding vertex positions. This method assumes linear behavior of the scalar function u on the edge inspected, which is supported by the piecewise linear assumption of the optimization. After all three edges are processed, there must be either 0 or 2 intersections because

   (a) either all three u values fall below or above u_c, in which case, there are no intersections; or

   (b) one falls below and the other two fall above, in which case there must be exactly two intersections; or

   (c) one falls above and the other two fall below, which results in the same situation as the previous case.

1. There is one exact match and it must be part of the contour. Examine the edge opposite to the corner with the exact match.

   (a) If the edge provides an intersection, the contour is the line segment defined by the exact match corner and the intersection.

(b) Otherwise, the exact match corner is the contour, which results in a degenerate line element for this triangle.

2. There are two exact matches, then the edge determined by these matches is the line segment generated.

3. There are three exact matches, then every point on the triangle is part of the contour. As such, there is no unambiguous way to form a line element as the contour for this triangle. This case is simply discarded. The reason is that having three exact matches means the triangle in the parametric space assumes a degenerate shape as its three edges overlap, which does not happen very often since the optimization seeks to minimize the orientation energy.

A similar contouring process can be applied to the v values as well. The results of the contouring process are shown in Figure 4.4.16, for a sphere, and for a Gabor surface. Note the regularity of the quadrangulation and the confinement of singularities to appropriate locations on the surface.

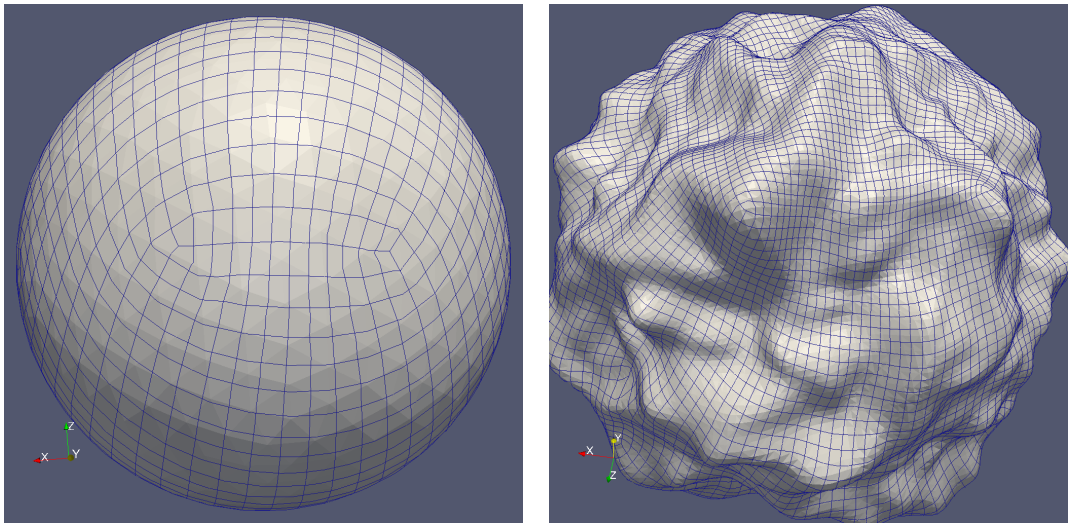


Figure 4.4.16: Grid-Like Line Structure

This chapter presented improvements based on the data analysis result obtained from previous user studies. In particular, the reason for the loss of effectiveness of line texture was analyzed and corresponding solution was proposed as well as the rationale for doing so. Next the advantages and disadvantages of principal curvature directions were discussed, leading to the design of a method based on principal curvature directions while avoiding its shortcomings. The decision to base this method on polygonal surfaces was also made since abundant existing research are available, thus

allowing for the design of an effective approach. Specifically, mesh saliency [56] was employed to identify important locations on a visualization target. A 4-way rotational symmetry field [70] is then generated based on guiding directions at these locations such as the principal curvature directions. After this, a globally smooth 2D parameterization [7] is formed that optimally follows the generated cross field. In the end, a Marching Cubes inspired algorithm was used to construct two sets of lines from the parameterization, effectively forming a grid-like uniform covering for the object examined. This method has the benefit to handle general shapes encountered in complex visualization, a subject treated next.

Chapter 5

Discussion

Chapter 4 presented details of a pipeline that is able to generate grid-like line patterns for general shapes as long as they can be described as triangle meshes having a 2-manifold topological structure. It also showed the results for a spherical surface and one of the gabor surfaces used in the user study discussed in Chapter 3. This chapter demonstrates more results tested on complex surfaces, extracted from real volume datasets, to show the method in practice. It also reveals places needing improvements, thus providing information to guide future research.

5.1 Results for Medical Volume Datasets

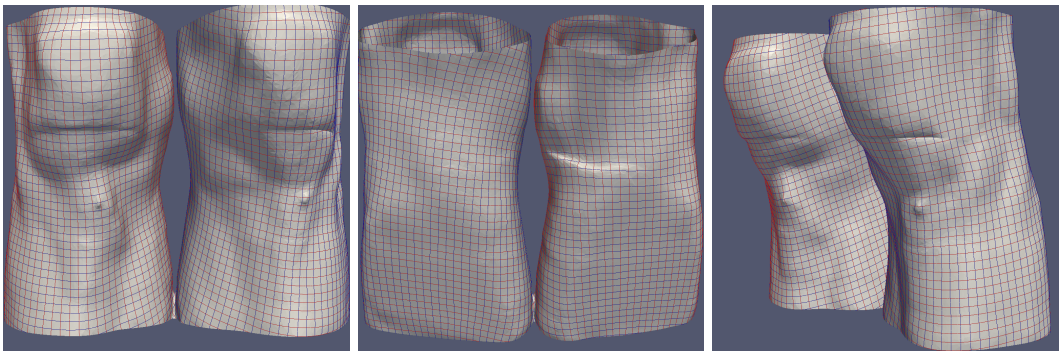


Figure 5.1.1: Knee Skin

Figure 5.1.1 shows several screen shots of the skin extracted from a knee volume dataset obtained from The Volume Library [1]. The red lines are constructed by regularly sampling in the

u direction of the parametric space and the blue lines in the v direction. Note how the skin of two legs appears to be connected at the lower part of the mesh as illustrated in Figure 5.1.2. This

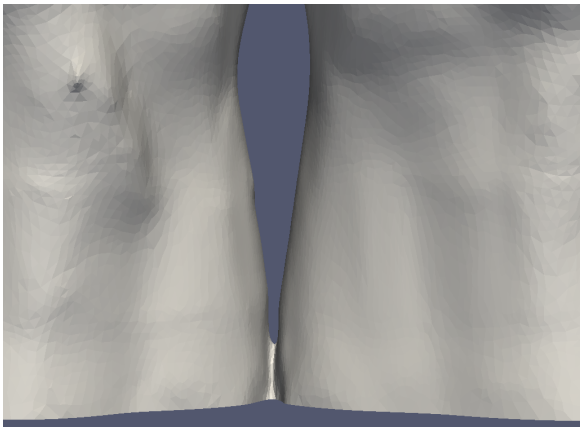
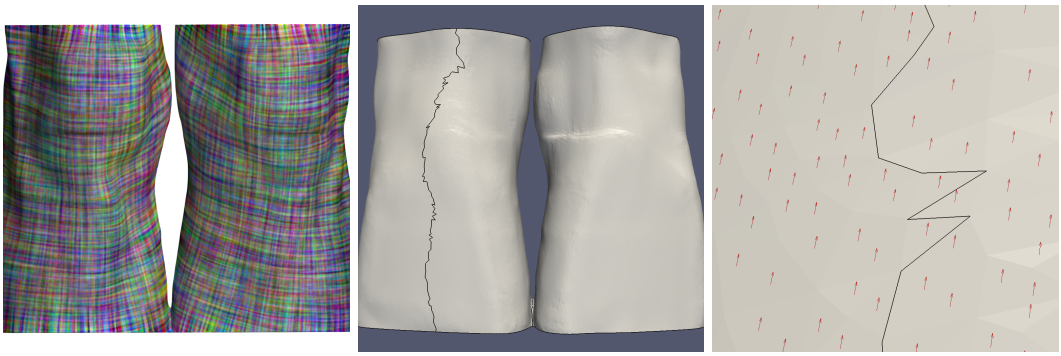


Figure 5.1.2: Knee Lower Part Connected

is because the surface extraction algorithm was not able to achieve a proper segmentation. This alters the topology of the extracted mesh in two ways. First, it generates one mesh instead of two disconnected meshes. Second, the resulting mesh does not have a cylindrical topology, making it harder to generate a proper grid structure covering the mesh. Nonetheless, as illustrated in Figure 5.1.1, the method successfully produces a uniformly structured grid texture. Figure 5.1.3a



(a) 4-RoSy Field (b) Knee Cut Graph (c) 1st RoSy Member Vector

Figure 5.1.3: Statistics of Knee Skin Parameterization

illustrates the 4-RoSy field used to guide the parameterization, while the black line segments in Figure 5.1.3b show the cut graph. Note the cut graph contains all of the boundary edges of the mesh. This is so because no boundary edge is shared by a pair of adjacent triangles. As such, they must be in the cut graph initially. Moreover, the simplification process cannot remove a boundary

edge since a mesh boundary forms a loop and no loops are added or removed by simplification. Figure 5.1.3c displays the first member vector of the per face 4-RoSy element after consistency propagation. The top row of Figure 5.1.4 indicates the positions of the singularities on the mesh, which are connected by the cut graph (black lines). Note that the singularities are placed at the artificial connection part, which is an ideal place for them, since they tend to create quadrilaterals with non-orthogonal edges, which would have caused visible degradation of the perpendicularity of the grid-like lines if they had been located elsewhere. The bottom image in Figure 5.1.4 presents a zoomed-in view to show the lines around singularities.

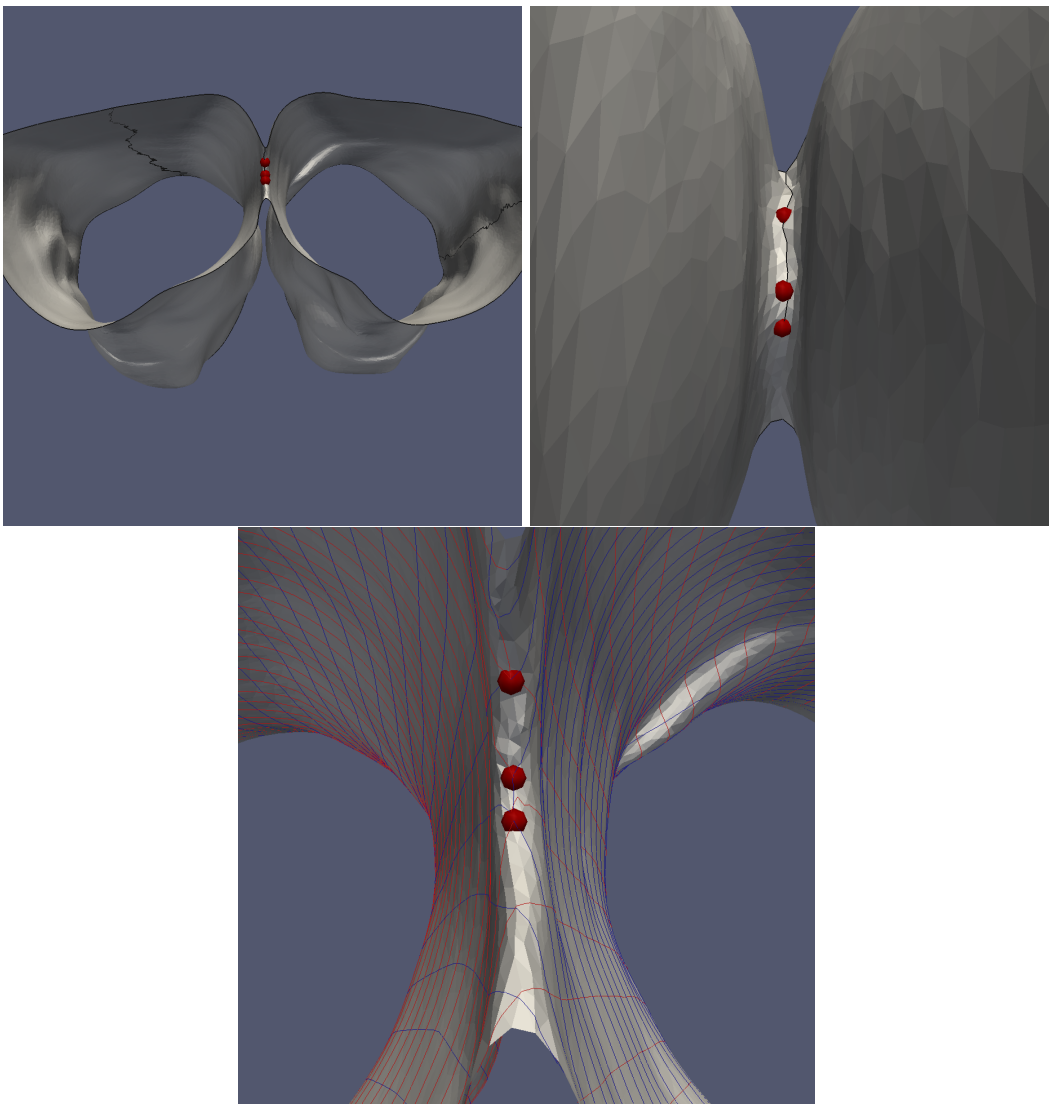


Figure 5.1.4: Knee Singularities

The following figures show the skin of a male human abdomen extracted from a volume dataset found on The Volume Visualization Organization [2]. Figure 5.1.8 and 5.1.9 give a front and back view of the abdomen skin, showing that the line structure captures the large scale features of the body. Figures 5.1.10 shows a side view of the skin and Figure 5.1.11 a zoomed-in view on irregular boundaries of the mesh. These artificial boundaries are generated by the mesh extraction algorithm, as a result of the sampling process that failed to capture the entire skin structure. The grid generation method successfully deals with these irregularities, which is an important property, since this is a phenomenon that commonly occurs when processing real volume datasets. Moreover, the grid-like lines generated on different sides of the irregular boundaries present a trend of continuity that can be utilized by human brain to virtually fill in the hole or connect the boundaries, thus creating a cognitive suggestion that the missing parts should be present. This property of the algorithm will most probably prove to be an important feature of the line structure in enhancing human understanding of complex shapes. Figure 5.1.5 displays the structure of the constructed lines at a less regular region of the mesh where most of singularities reside. The method still generates reasonably well-structured grid-like lines, which demonstrates its ability to work with irregular features in the mesh.

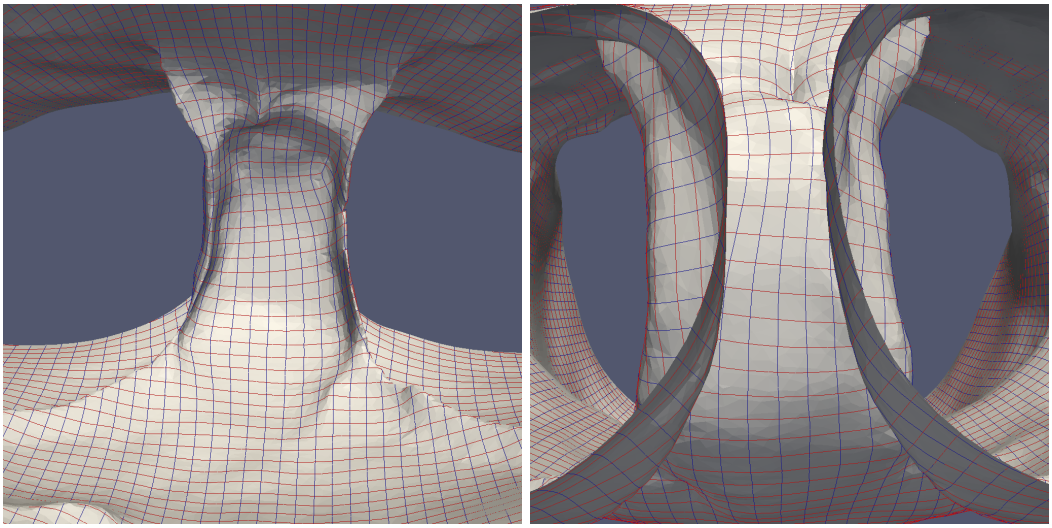


Figure 5.1.5: Abdomen Skin Sharp Features and Irregularities

Figure 5.1.12 and 5.1.13 show the bone structure extracted from the same volume dataset as the previous abdomen skin surface. The figure again demonstrates the ability of our method to work with irregular shapes encountered in real volume datasets. Figure 5.1.6 provides enlarged views of some irregular regions on the structure to show the performance of the method.

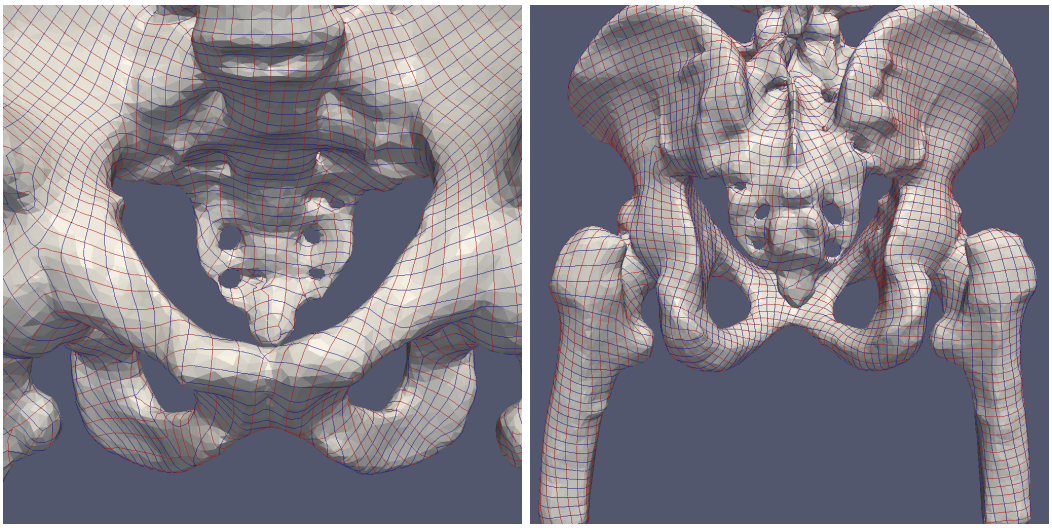


Figure 5.1.6: Bone Enlarged View

The primary goal of this research is to explore techniques assisting human perception in a complex visualization environment involving multiple (mostly two) targets. To demonstrate the grid

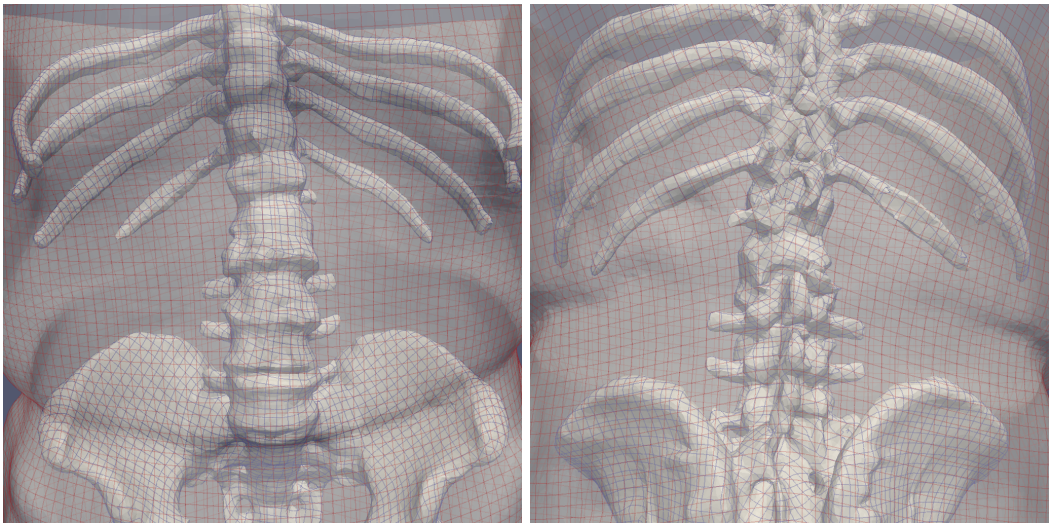


Figure 5.1.7: Abdomen Skin and Bone Enlarged View

lines are helpful in these contexts, Figure 5.1.14 and 5.1.15 show overlapped renderings of the skin

and bone with transparency assigned to the skin. The lines on both surfaces use transparency, with optimal opacity values identified by the user study discussed in Chapter 3. Figure 5.1.7 displays enlarged views of the spine. It can be seen that both structures are better visualized with the added lines describing their geometric structure. Based on our experience, such effects will be even stronger in a stereoscopic display environment.

The results on medical volume datasets demonstrate that the method discussed in Chapter 4 has the ability to process complex surfaces encountered in real applications. There are also possible improvements that can be made based on our observation in practice, which are discussed next.

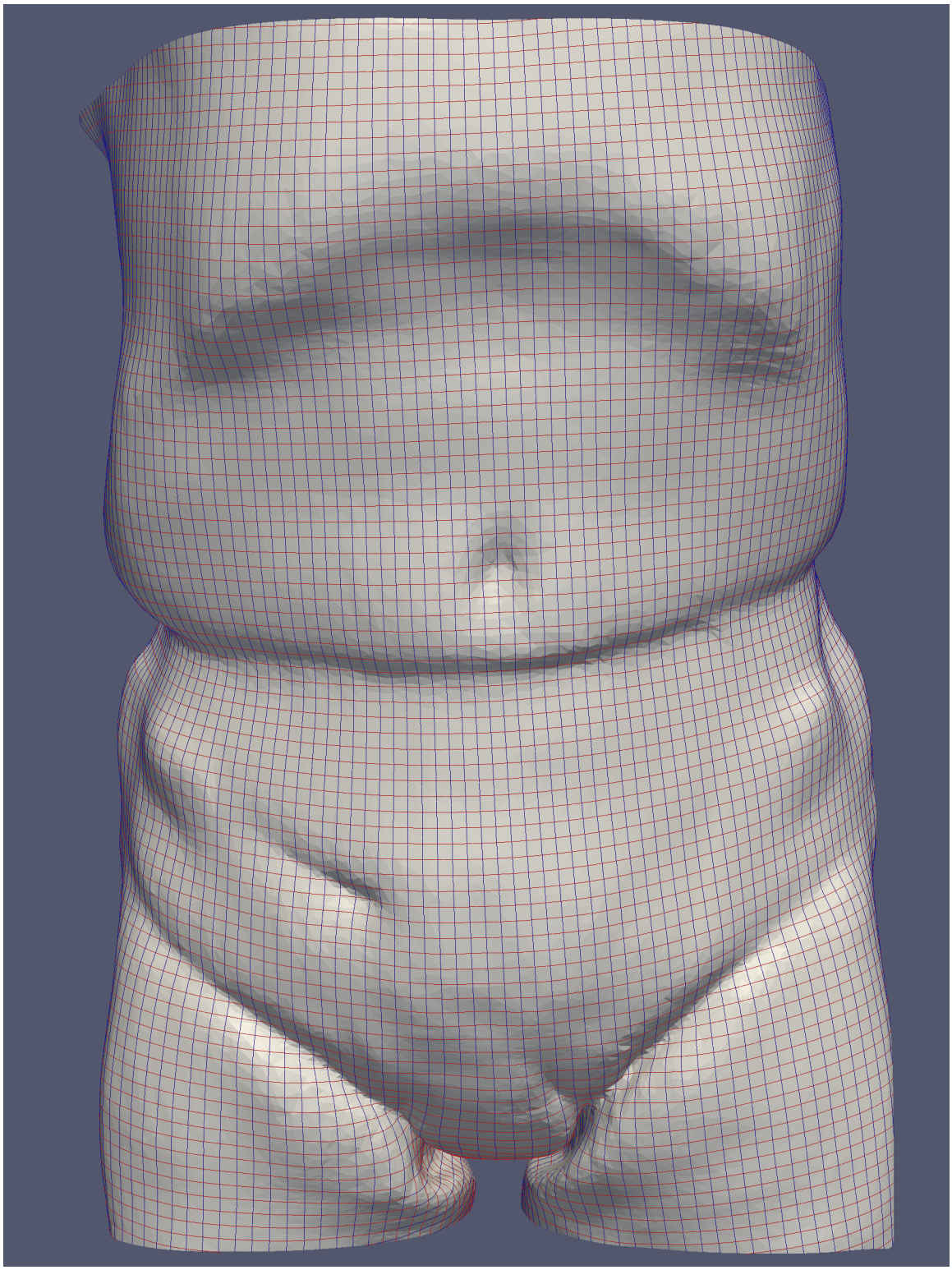


Figure 5.1.8: Abdomen Front

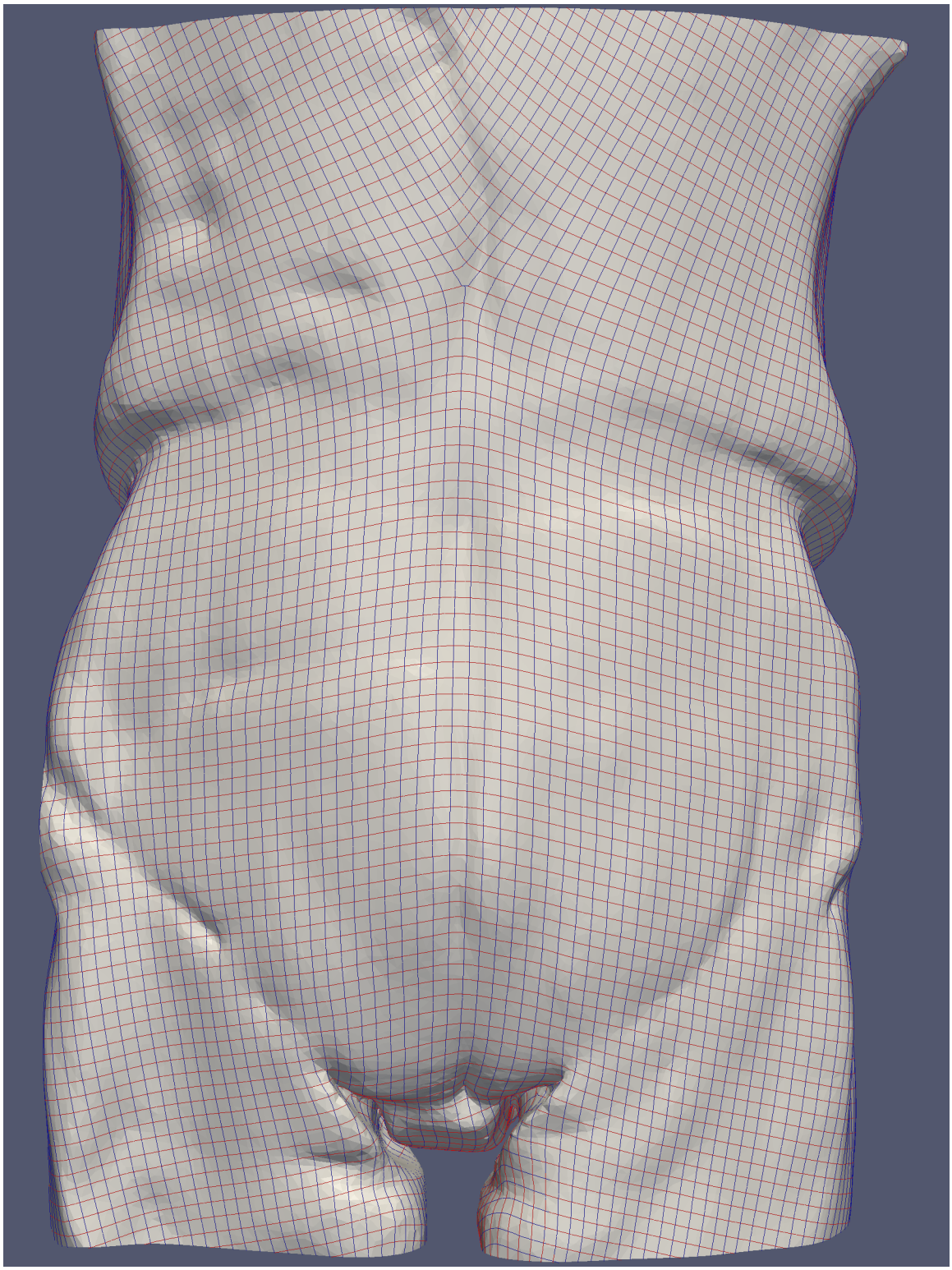


Figure 5.1.9: Abdomen Back

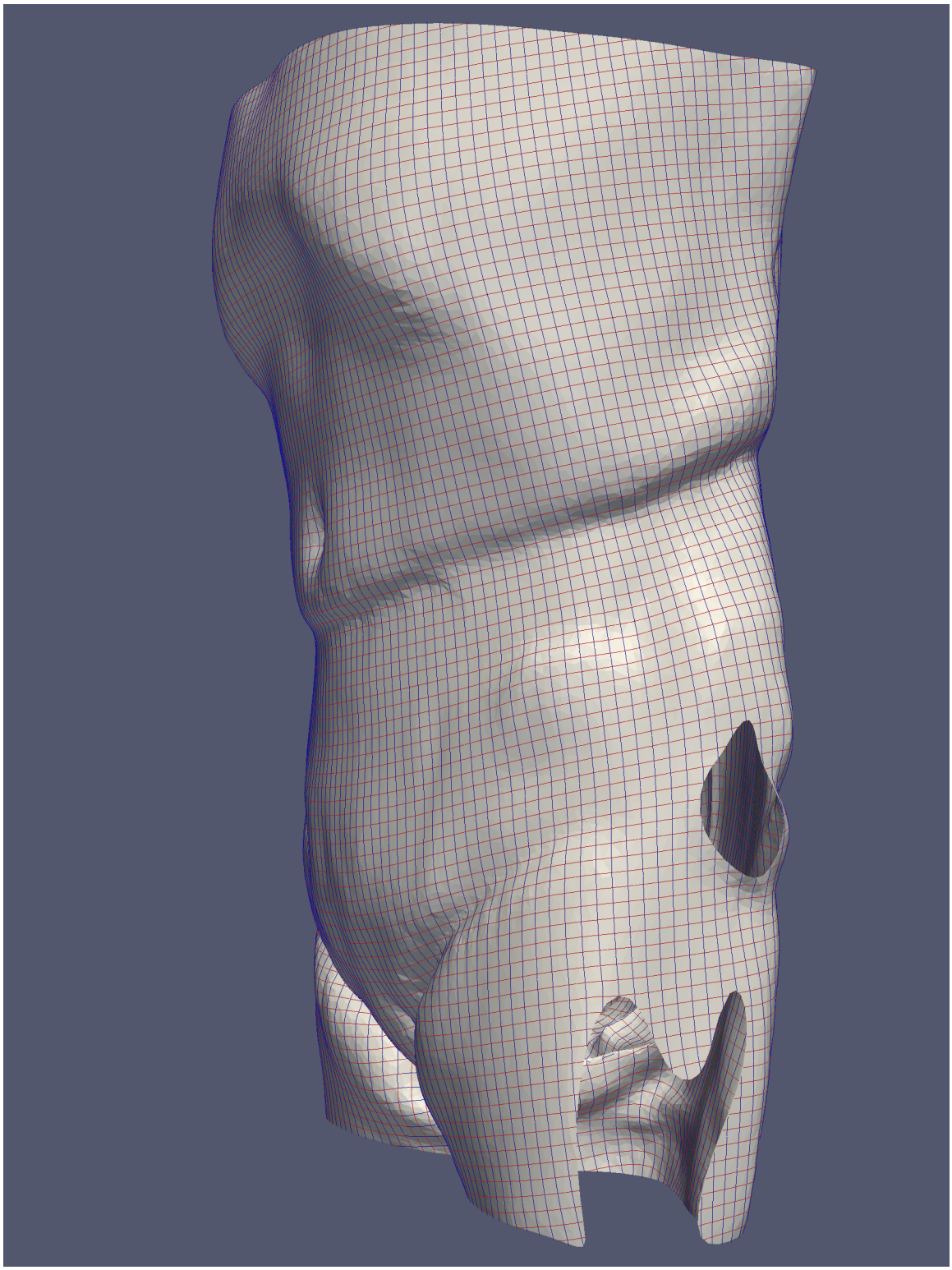


Figure 5.1.10: Abdomen Side View

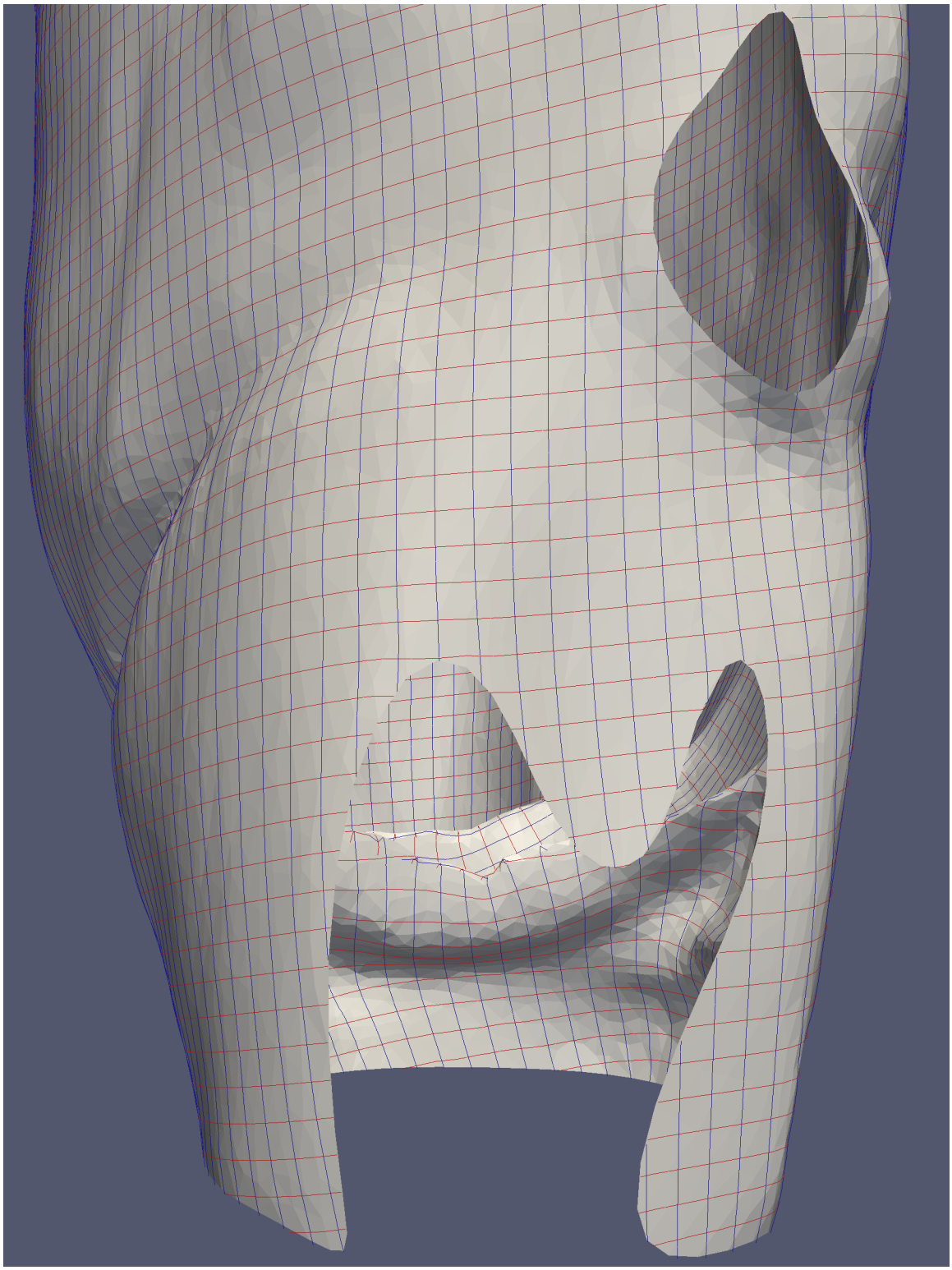


Figure 5.1.11: Abdomen Side Zoom In

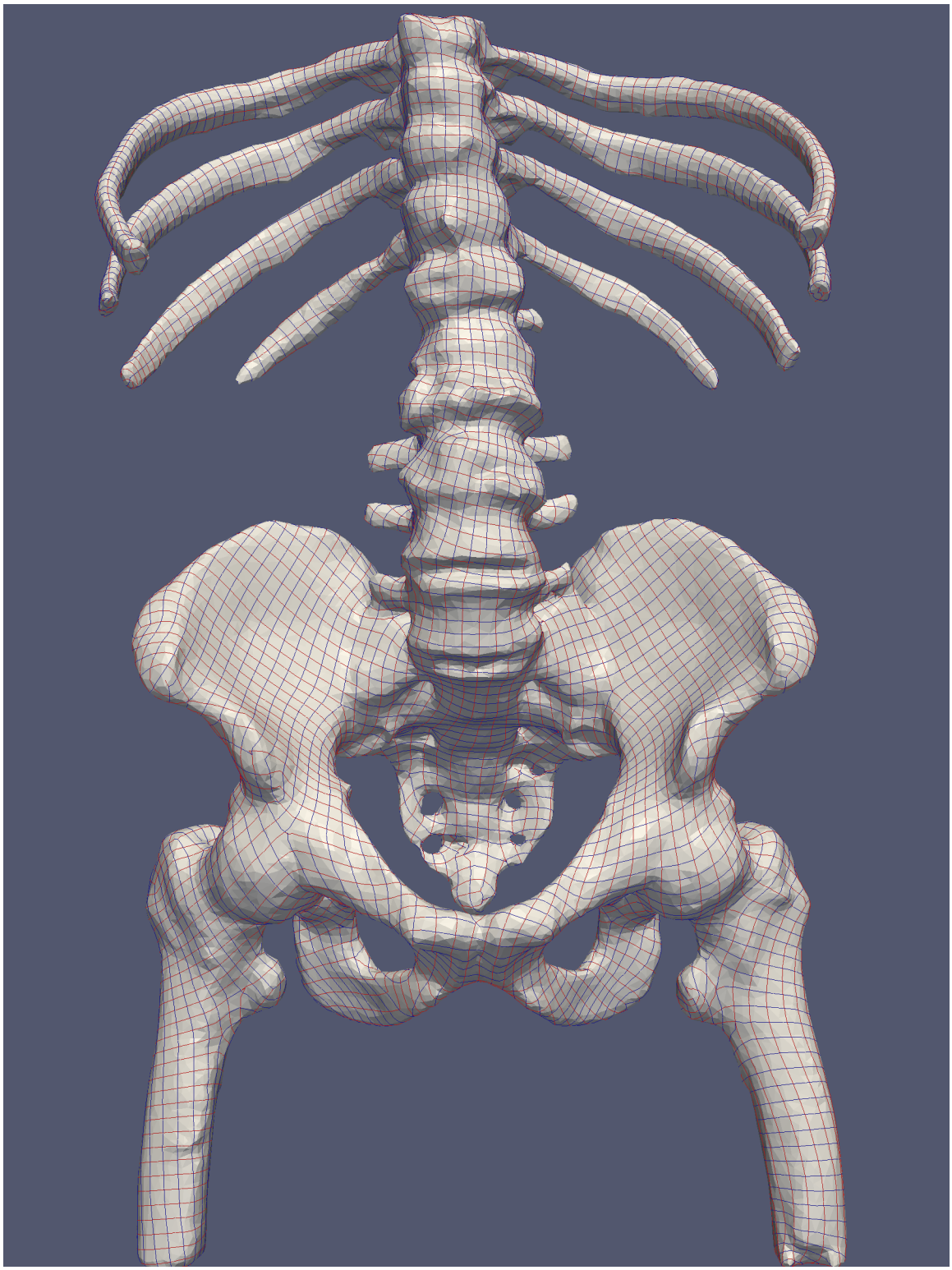


Figure 5.1.12: Bone Front

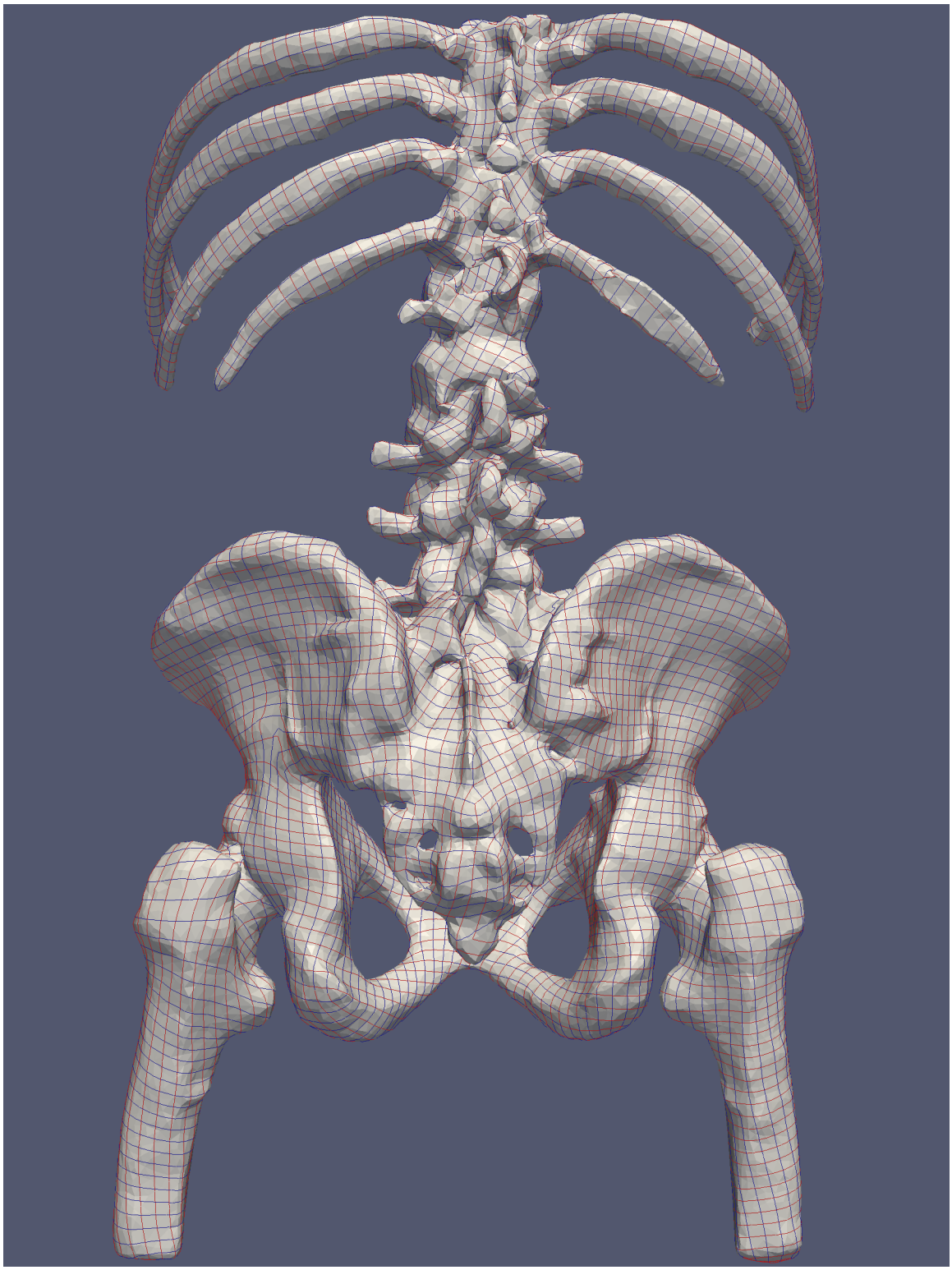


Figure 5.1.13: Bone Back

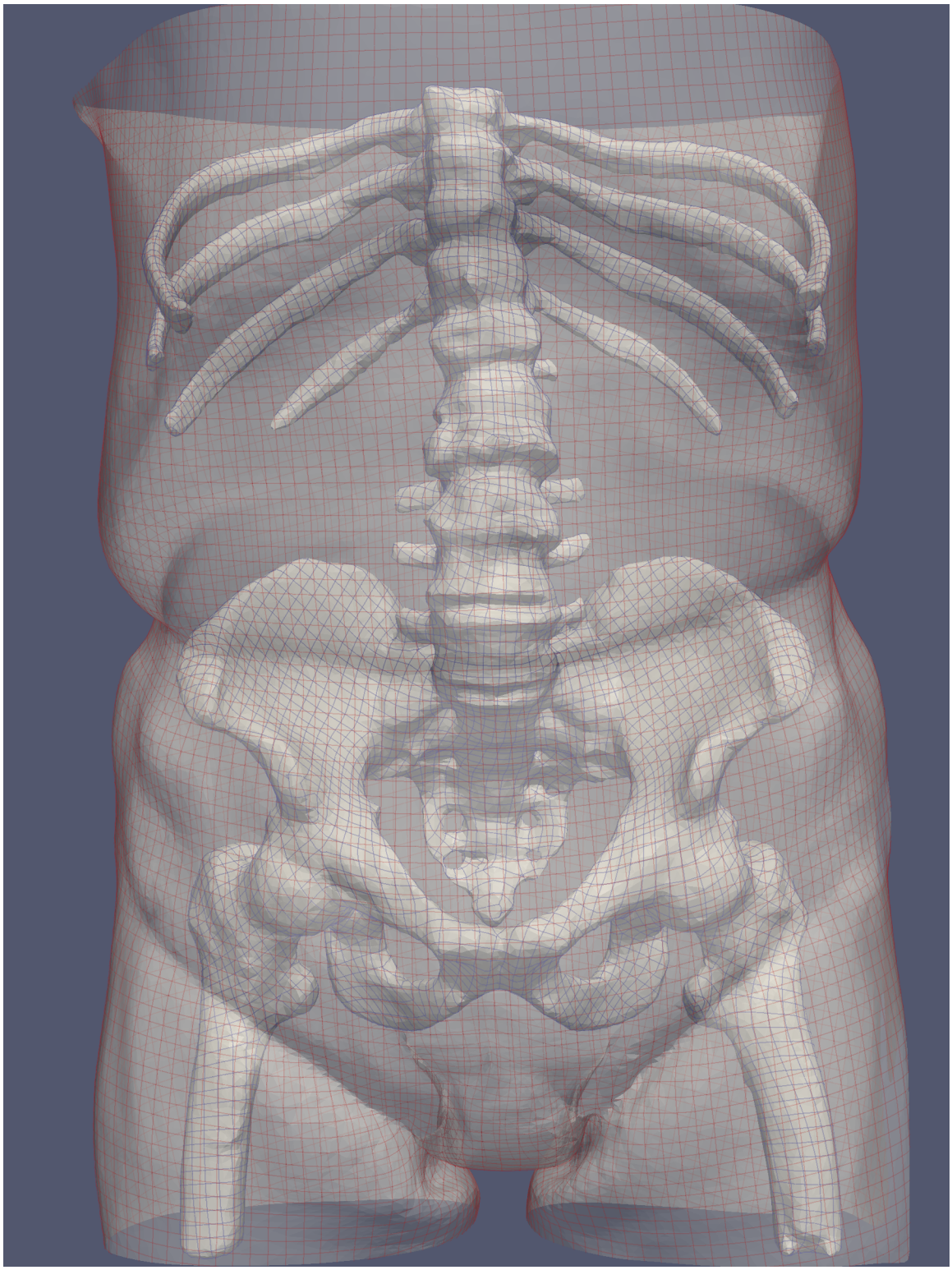


Figure 5.1.14: Abdomen Skin and Bone Front

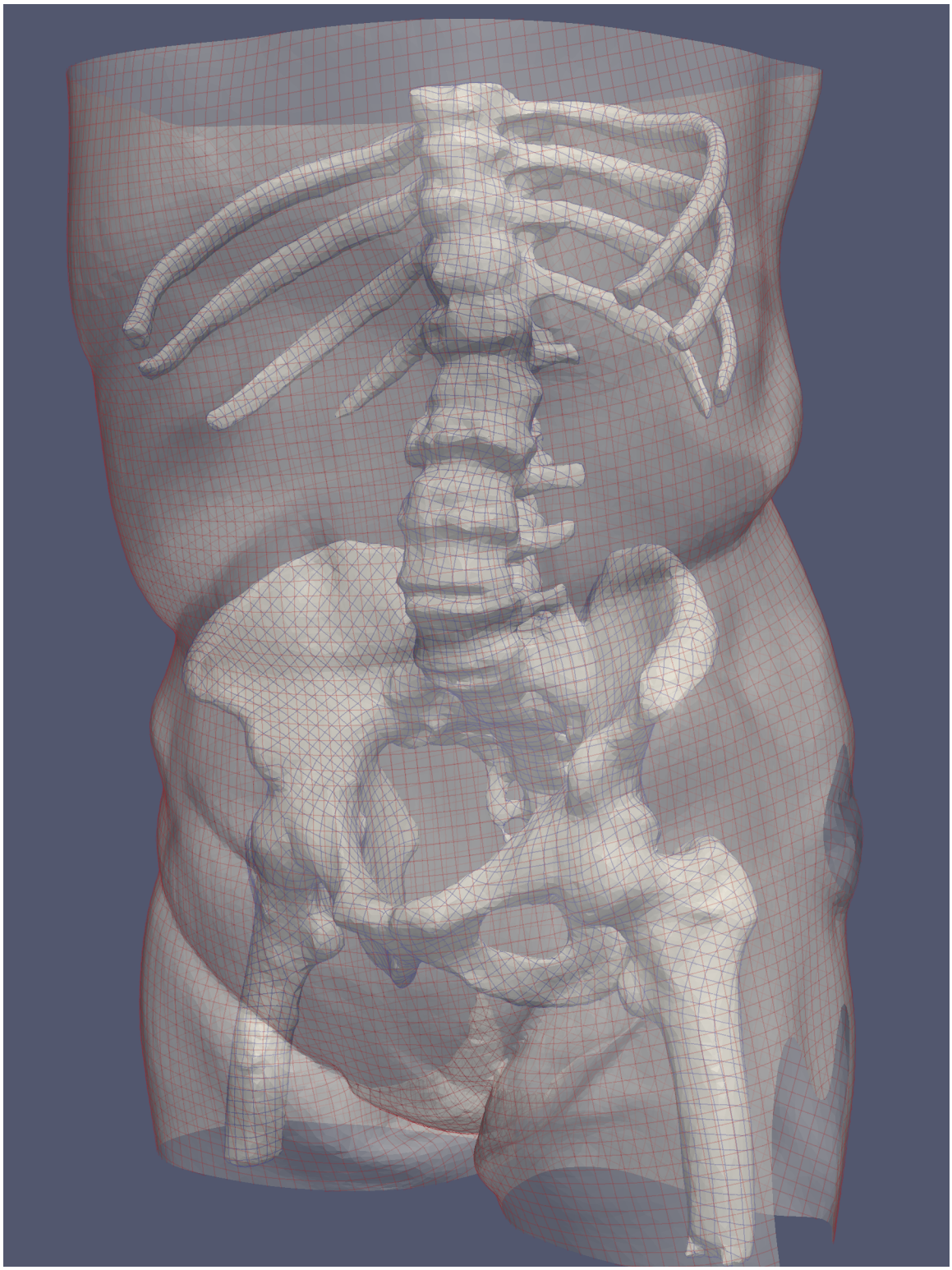


Figure 5.1.15: Abdomen Skin and Bone Side

5.2 Future Directions

5.2.1 Mesh Simplification

The optimization problem defined in Section 4.4 requires the assignment of a 2D coordinate to each vertex in the mesh. For complicated meshes, with a large number of vertices, the optimization becomes computationally impractical to solve since there are too many sub-problems to be solved for each optimization. For example, the knee skin as illustrated in Figure 5.1.1 originally contains about 500K vertices, which roughly demands 1 million continuous variables for the parameterization. In our experience, the solver is not able to produce a result within reasonable amount of time, making the problem effectively unsolvable. To cope with this complexity, the mesh must be simplified before it can be parameterized. By experiment, we found that 50K triangles gives a good balance between computability and feature preservation. Decimation of triangle meshes has long been a research focus in computer graphics, with several algorithms available. Due to the following concerns, however, human intervention is desirable in producing the simplified mesh for parameterization.

1. The parameterization requires a 2-manifold topology, which is not guaranteed by the decimation process. As such, it is crucial to eliminate non-manifold vertices and non-manifold edges and triangles containing such entities, a process creating holes or modifying mesh boundaries, thus requiring human inspection to ensure feature preservation.

2. Simplification is essentially a modification process. In this process, human judgment is needed to preserve important mesh features, since their automatic detection is difficult.

3. Simplification may introduce very sharp and thin features, which are undesirable. Consequently, mesh editing and smoothing are usually necessary as additional steps, which might cause the loss of features, thus requiring human examination.

5.2.2 Singularities Reduction

For complicated meshes, there can be many singularities generated by the 4-RoSy field algorithm discussed in Section 4.3. For example, the abdomen bone structure presented in Section 5.1 has 651 singularities as illustrated in Figure 5.2.1.

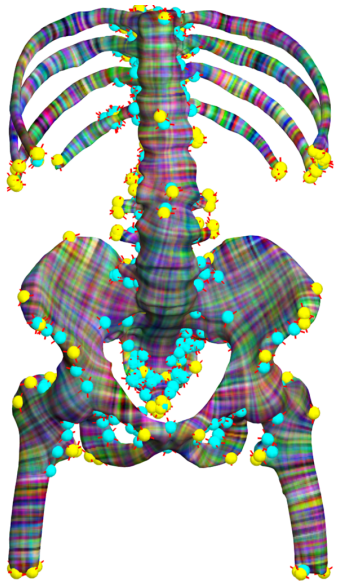


Figure 5.2.1: Abdomen Bone Singularities

Having many singularities significantly increases the computational complexity of the optimization. This is so because singularities must possess integer values in the parametric space, which requires the creation and solution of more sub-problems seeking integer solutions. A large number of singularities also degrades the quality of the generated grid lines because parameterization for regions around singularities tends to create quadrilaterals with non-orthogonal boundaries. Consequently, reducing the number of singularities is beneficial to the overall process. Due to the topological constraints of the mesh, singularities can only be eliminated through paired cancellation, i.e., one maximum or minimum being canceled with a saddle. Currently, only manual editing is supported, which is a slow and tedious process. Therefore, automatic pairing and cancellation would be a promising direction.

In summary, this chapter presented results obtained for complex surfaces extracted from medical volume datasets and demonstrated the performance of our method on them. The generated grid-like lines can be utilized by the renderer developed in Section 3.1, making it ready for further evaluation of the effectiveness of these grid lines in enhancing human perception for complex shapes in real volume datasets.

Chapter 6

Conclusion

The goal of this research is to help enhance perceptions of shapes in a volume rendering process. Based on existing research, we rely on the application of grid-like line textures covering shapes in a consistent manner to realize such enhancement.

Specifically, this thesis discussed the concept of volume data. It also covered the theory of volume rendering, its advantages and disadvantages, and the existing implementation methods, primarily GPU-based due to the efficiency requirement. It examined various descriptive line elements with the goal to convey shape information of covered geometry. House's group [4, 5, 6, 31] studied grid-like line texture of carefully designed overlapping terrain surfaces, finding that such texture is one of the most effective in terms of helping humans perceive the terrain shape. Since line textures enhance human perception for surface-based graphics, we extended it to volume visualization. The work completed confirmed, through a user study and statistical analyses, that these line elements work in a volume rendering process. The experiments indicated that a projected grid texture loses its effectiveness when the viewer is looking at the shape from a perpendicular perspective. Because of the spatial arrangement, however, this case was intentionally avoided in previous research by House's group. This problem was addressed by line elements that bear intrinsic geometrical information. Of particular interest is the principal curvature direction guided line texture as their effectiveness has been verified by previous research [35].

Nonetheless, principal curvature directions also have disadvantages, such as being sensitive to noise. Realizing this, we utilized principal curvature directions to generate a grid-like line structure that is guided by the former. Specifically, mesh saliency was used to mark the importance of all

vertices on a polygonal surface which represents the visualization target of interest. This highlights a few important locations whose principal curvatures can be computed and used to generate a globally smooth cross field that assigns, for each mesh vertex, a set of 4 mutually orthogonal directions identifying important directions at the same location. Such field was then utilized to form a global 2D parameterization for the entire surface, one that minimized the least square error between the gradients of the parameterization and the cross field. This optimal parameterization formed the foundation for a Marching Cubes inspired contouring algorithm that eventually constructed two sets of lines resembling a grid-like line structure uniformly covering the visualization target.

The improved generation method has two major desirable advantages. First, it works with general shapes as long as such shapes can be expressed as 2-manifold triangle meshes, a property making the method applicable to a wide range of visualization targets. Second, it allows humans to control both the important feature locations and guiding directions, an attribute we considered beneficial for perception based applications since subjective understanding usually requires fine tuning by each individual. The method is then tested on several surfaces extracted from real medical volume datasets, which showed its ability and revealed a few potential improvements as well, thus providing valuable guidance for future research.

In summary, this research verified that grid-like line texture helps humans to better understand shapes in a complex visualization environment. It also designed an effective method that generates such line structure for shapes encountered in real volume datasets, thus achieving the primary goal of providing assistance for perception based volume visualization.

Appendices

Appendix A Base Sphere Radii for Two Volume Objects Case

In the two-volume-object experiments, a pair of irregular shapes are needed with one embedded in the other. They are built from spheres and are required to have no intersections. The inner shape needs to be as large as possible so that it looks reasonably large under the perspective projection in rendering. This requires a carefully chosen sphere radii as the starting shapes. The choice is 10 cm for the inner sphere and 13 cm for the outer one, for a virtual screen of size 29.88 cm.

The choice of 10 is based on Equation 3.2.5, which shows that the σ values of the Gabor functions of different levels are approximately 2.88, 0.96 and 0.32. These σ values ensure that the Gaussian components are smooth at the largest level and are not too sharp at the smallest level, important because sharp features produce significant variations of normal directions within a small region on a Gabor surface due to its high curvature, a situation that is difficult to capture for humans in nature. The choice of 13 is based on the requirement that the outer shape needs to barely contain the inner shape.

The following analysis shows the reasoning. The global maximum of the Gabor function is controlled by the global maximum of the Gaussian component and by the amplitude A. Given the way the Gabor functions are generated, it is possible for three Gabor functions to stack in one radial direction, with each one coming from one of the three different levels. The Gaussian function is bounded above by the constant 1, meaning the global maximum for the Gabor function is therefore bounded above by its amplitude parameter A which can be expressed with respect to the sphere radius r by

$$\begin{aligned} A &= 0.1T \\ T &= \tfrac{10}{3}\sigma \\ \hat{\sigma} &= \tfrac{r}{2}\tan(30^\circ). \end{aligned} \tag{A.1}$$

These equations indicate that the global maximum of the Gabor function is bounded above by $\sigma/3$. Moreover, the σ parameter is drawn from a Gaussian distribution of a mean value $\hat{\sigma}$ and a standard deviation $0.08\hat{\sigma}$, meaning that the σ value for the Gabor function is bounded above by $\hat{\sigma} + 3 \times 0.08\hat{\sigma}$ with high probability. Therefore, the maximum of the Gabor function is bounded above by

$$\frac{1}{3}\sigma < \frac{1}{3} \times (1 + 3 \times 0.08)\hat{\sigma} < \frac{1}{3} \times \frac{5}{4} \times \frac{r}{2} \times \frac{\sqrt{3}}{3} = \frac{5\sqrt{3}}{72} r \approx 0.1293r.$$

The scale factor of σ between succeeding levels of Gabor functions is chosen to be $1/3$. Therefore, the total amount of displacement that can possibly be caused by three stacked Gabor functions that come from three different levels is bounded above by

$$(1+\frac{1}{3}+\frac{1}{9})\times\frac{5\sqrt{3}}{72}r=\frac{65\sqrt{3}}{648}r\approx 0.1737r.$$

This equation means that the expected largest displacement caused by Gabor surfaces on the inner sphere surface is approximately 1.737, meaning that the far reach of the inner shape is $10+1.737$. To see if there are any intersections between the two shapes, however, it is necessary to know how much the outer shape can deform inside. The inward deformation is regulated by the global minimum of the Gabor function which is reached at the same point of the global minimum of the cosine component in the Gabor function as defined in Equation 3.2.1. Mathematically, the cosine function reaches its global minimum at an infinite number of locations. However, the Gaussian component modulates the cosine function in such a way that the local minimum closest to the origin is the global minimum of the Gabor function. As such, the global minimum of the Gabor function appears at $T/2$ or at $5\sigma/3$ based on Equation A.1. The magnitude of the Gaussian component at $(5\sigma/3, 0)$ is

$$e^{-\frac{\left(\frac{5}{3}\sigma\right)^2}{2\sigma^2}}=e^{-\frac{25}{18}}\approx 0.2494.$$

Following the same logic, the largest displacement that can be caused by any three stacked Gabor functions is bounded below by

$$-e^{-\frac{25}{18}}\frac{65\sqrt{3}}{648}r\approx -0.04333r.$$

For the external sphere radius 13, this equation gives -0.563, meaning that the outer shape can deform a maximum of 0.563 towards the origin following any radial direction. The closest points between the two shapes, therefore, have radial distances of $10+1.737$ versus $13-0.563$ statistically. This spread gives a difference of 0.7 which works as an appropriate margin between them. Thus, 10 and 13 satisfy the needs of this research.

However, the guarantee of having no intersections is only statistically accurate in two aspects. First, the reasoning relies on the 3σ-rule of the Gaussian distribution. This property only means that the chance is small of drawing a value from a Gaussian distribution that is farther away from the mean value by 3σ. However, this chance is never denied. Second, the reasoning is based

on the assumption that the largest offset caused by the Gabor surfaces is reached by stacking three Gabor functions of different levels on one radial direction and that nothing else will cause further displacement in this direction. However, the Poisson sampling method used here guarantees that only the center of any Gabor function will not fall into the span of any other Gabor functions in the same magnitude level. Their spans, however, can still overlap, meaning that there is a small chance that their spans can violate the above assumption. The Poisson sampling does guarantee, though, that there cannot be more than a few such Gabor surfaces that overlap for any given point on the sphere surface. Because of this fact combined with the safety buffer of 0.7 between the two shapes, this research has never observed any such randomly generated shapes to have intersections; although several cases have come close, they have not affected the research reported here.

Bibliography

[1] The volume library. `http://www9.informatik.uni-erlangen.de/External/vollib/`.

[2] The volume visualization organization. `http://www.volvis.org/`.

[3] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[4] A. Bair and D. House. Grid with a view: Optimal texturing for perception of layered surface shape. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1656 –1663, nov.-dec. 2007.

[5] A. Bair, D.H. House, and C. Ware. Texturing of layered surfaces for optimal viewing. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1125 –1132, sept.-oct. 2006.

[6] Alethea S. Bair, Donald H. House, and Colin Ware. Factors influencing the choice of projection textures for displaying layered surfaces. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*, APGV '09, pages 101–108, New York, NY, USA, 2009. ACM.

[7] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 77:1–77:10, New York, NY, USA, 2009. ACM.

[8] David Bommes, Henrik Zimmer, and Leif Kobbelt. Practical mixed-integer optimization for geometry processing. In *Curves and Surfaces*, pages 193–206. Springer, 2012.

[9] FW Campbell and DG Green. Optical and retinal factors affecting visual resolution. *The Journal of Physiology*, 181(3):576, 1965.

[10] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679 –698, nov. 1986.

[11] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350 – 355, 1978.

[12] Evgeni V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, Institute for High Energy Physics, 1995.

[13] H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford, and B.C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical physics*, 15:320, 1988.

[14] D. Cohen and Z. Sheffer. Proximity clouds - an acceleration technique for 3d grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[15] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *Communications of the ACM*, 55(1):107–115, January 2012.

[16] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering, feb 2009. to appear.

[17] John G. Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Research*, 20(10):847 – 856, 1980.

[18] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, July 2003.

[19] C.A. Dietrich, C.E. Scheidegger, J.L.D. Comba, L.P. Nedel, and C.T. Silva. Marching cubes without skinny triangles. *Computing in Science Engineering*, 11(2):82 –87, march-april 2009.

[20] Debra Dooley and Michael F. Cohen. Automatic illustration of 3d geometric models: lines. *SIGGRAPH Comput. Graph.*, 24(2):77–82, February 1990.

[21] Christopher Dyken, Gernot Ziegler, Christian Theobalt, and Hans-Peter Seidel. High-speed marching cubes using histopyramids. *Computer Graphics Forum*, 27(8):2028–2039, 2008.

[22] Gershon Elber and Elaine Cohen. Hidden curve removal for free form surfaces. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 95–104, New York, NY, USA, 1990. ACM.

[23] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, and Daniel Weiskopf. Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA, 2004. ACM.

[24] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pages 9–16, New York, NY, USA, 2001. ACM.

[25] Ergun Akleman *et al.* Topmod download page. `http://www.viz.tamu.edu/faculty/ergun/research/topology/download.html`.

[26] Michael S Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[27] Sarah Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In William Wells, Alan Colchester, and Scott Delp, editors, *Medical Image Computing and Computer-Assisted Intervention MICCAI98*, volume 1496 of *Lecture Notes in Computer Science*, pages 888–898. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0056277.

[28] F. Goetz, T. Junklewitz, and G. Domik. Real-time marching cubes on the vertex shader. In *Proceedings of Eurographics*, volume 2005, 2005.

[29] Khronos Group. Khronos group home page. `http://www.khronos.org/`.

[30] Kai Hormann, Bruno Lévy, and Alla Sheffer. Mesh Parameterization: Theory and Practice, 2007. This document is the support of a course given at SIGGRAPH 2007.

[31] D.H. House, A.S. Bair, and C. Ware. An approach to the perceptual optimization of complex visualizations. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):509 –521, july-aug. 2006.

[32] Jin Huang, Muyang Zhang, Jin Ma, Xinguo Liu, Leif Kobbelt, and Hujun Bao. Spectral quadrangulation with orientation and alignment control. In *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pages 147:1–147:9, New York, NY, USA, 2008. ACM.

[33] Kitware Inc. Paraview home page. `http://www.paraview.org/`.

[34] V. Interrante, H. Fuchs, and S.M. Pizer. Conveying the 3d shape of smoothly curving transparent surfaces via texture. *Visualization and Computer Graphics, IEEE Transactions on*, 3(2):98 –117, apr-jun 1997.

[35] V. Interrante and S. Kim. Investigating the effect of texture orientation on the perception of 3d shape. In *Human Vision and Electronic Imaging VI*, volume 4299, pages 330–339, 2001.

[36] Victoria Interrante. Illustrating surface shape in volume data via principal direction-driven 3d line integral convolution. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 109–116, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[37] Victoria Interrante, Henry Fuchs, and Stephen Pizer. Enhancing transparent skin surfaces with ridge and valley lines. In *Proceedings of the 6th conference on Visualization '95*, VIS '95, pages 52–, Washington, DC, USA, 1995. IEEE Computer Society.

[38] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(11):1254–1259, 1998.

[39] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of 8th Eurographics Workshop in Scientific Computing*, pages 45–55, April 1997.

[40] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 339–346, New York, NY, USA, 2002. ACM.

[41] Tao Ju, Scott Schaefer, and Joe Warren. Convex contouring of volumetric data. *The Visual Computer*, 19:513–525, 2003. 10.1007/s00371-003-0216-0.

[42] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[43] R. Kahler, M. Simon, and H.-C. Hege. Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):341 – 351, july-sept. 2003.

[44] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover-surface parameterization using branched coverings. In *Computer Graphics Forum*, volume 26, pages 375–384. Wiley Online Library, 2007.

[45] Khronos OpenCL Working Group. *The OpenCL Specification, version 1.0.29*, 8 December 2008.

[46] T. Klein, S. Stegmaier, and T. Ertl. Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 186 – 195, oct. 2004.

[47] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 57–66, New York, NY, USA, 2001. ACM.

[48] C. Koch and S. Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. *Hum Neurobiol*, 4(4):219–27, 1985.

[49] Jan J. Koenderink. *Solid Shape*. Artificial Intelligence Series. MIT Press, Cambridge, MA, USA, 1990.

[50] J.J. Koenderink, A.J. Van Doorn, et al. The shape of smooth objects and the way contours end. *Perception*, 11(2):129–137, 1982.

[51] Michael Kolomenkin, Ilan Shimshoni, and Ayellet Tal. Demarcating curves for shape illustration. *ACM Trans. Graph.*, 27(5):157:1–157:9, December 2008.

[52] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458. ACM, 1994.

[53] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 55–63, New York, NY, USA, 2010. ACM.

[54] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, NPAR '00, pages 13–20, New York, NY, USA, 2000. ACM.

[55] Byeonghun Lee, Jihye Yun, Jinwook Seo, Byonghyo Shim, Yeong-Gil Shin, and Bohyoung Kim. Fast high-quality volume ray casting with virtual samplings. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1525–1532, November 2010.

[56] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. *ACM Trans. Graph.*, 24(3):659–666, July 2005.

[57] Marc Levoy. Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9(3):245–261, July 1990.

[58] T. Lewiner, H. Lopes, A.W. Vieira, and G. Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools*, 8(2):1–16, 2003.

[59] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[60] Samira Mahdizadeh. Texturing 3d curved surfaces according to principal curvature information using evenly spaced streamlines of arbitrary density. *Baha'i Institute for Higher Education*, B.S. Thesis, 2008.

[61] Benoit B. Mandelbrot. *The fractal geometry of nature*. W. H. Freeman, New York, 1983.

[62] B. Matérn et al. Spatial variation. stochastic models and their application to some problems in forest surveys and other sampling investigations. *Meddelanden fran statens Skogsforskningsinstitut*, 49(5), 1960.

[63] J.R. Movellan. Tutorial on gabor filters. `http://mplab.ucsd.edu/tutorials/gabor.pdf`.

[64] B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11:52–62, 1994. 10.1007/BF01900699.

[65] T.S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, 2006.

[66] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 83–91, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[67] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.

[68] H. Nyquist. Certain topics in telegraph transmission theory. *Proceedings of the IEEE*, 90(2):280 –305, feb 2002.

[69] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice–Hall, 1975.

[70] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3), July 2007.

[71] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[72] Randi J. Rost, Bill Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL Shading Language*. Addison-Wesley Professional, 3rd edition, 2009.

[73] Szymon Rusinkiewicz, Forrester Cole, Doug DeCarlo, and Adam Finkelstein. Line drawings from 3d models. In *ACM SIGGRAPH 2008 Courses*, SIGGRAPH '08, New York, NY, USA, 2008. ACM.

[74] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206, September 1990.

[75] J. Schreiner, C.E. Scheiclegger, and C.T. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1205 –1212, sept.-oct. 2006.

[76] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[77] P. Shirley and A. Tuchman. *A polygonal approximation to direct scalar volume rendering*, volume 24. ACM, 1990.

[78] Renben Shu, Chen Zhou, and Mohan S. Kankanhalli. Adaptive marching cubes. *THE VISUAL COMPUTER*, 11:202–217, 1995.

[79] E. Smistad, A.C. Elster, and F. Lindseth. Fast surface extraction and visualization of medical images using opencl and gpus. In *The Joint Workshop on High Performance and Distributed Computing for Medical Imaging (HP-MICCAI)*, 2011.

[80] Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 902–907, 1995.

[81] Anne M. Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97 – 136, 1980.

[82] Kouki Watanabe and Alexander G Belyaev. Detection of salient curvature features on polygonal surfaces. In *Computer Graphics Forum*, volume 20, pages 385–392. Wiley Online Library, 2001.

[83] C. Wheatstone. Contributions to the physiology of vision.–part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical transactions of the Royal Society of London*, 128:371–394, 1838.

[84] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[85] Mann-May Yau and Sargur N. Srihari. A hierarchical data structure for multidimensional digital images. *Commun. ACM*, 26(7):504–515, July 1983.

[86] Long Zhang, Ying He, Xuexiang Xie, and Wei Chen. Laplacian lines for real-time shape illustration. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 129–136, New York, NY, USA, 2009. ACM.

[87] G. Ziegler, A. Tevs, C. Theobalt, and H.P. Seidel. On-the-fly point clouds through histogram pyramids. In *Workshop on Vision, Modeling, and Visualization (VMV 2006)*, pages 137–144, 2006.